UNIVERSITÉ DE
SHERBROOKE

Faculté de Génie
Département de Génie Électrique et de Génie Informatique

RÉALISATION D'UN ASSISTANT MOBILE EXPÉRIMENTAL POUR USAGERS À BESOINS SPÉCIAUX

Mémoire de maîtrise ès sciences appliquées
Spécialité : génie électrique

Composition du jury

Daniel Dalle
Philippe Mabilleau
Bessam Abdulrazak

Alexandra FERRER

Sherbrooke (Québec), Canada

18 septembre 2008

TV-1905

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-49498-1
Our file Notre référence
ISBN: 978-0-494-49498-1

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

RÉSUMÉ

L'ONU estime à 650 millions le nombre de personnes à besoins spéciaux, soit presque 11% de la population mondiale. En Europe, on estime à 1 million le nombre de personnes ne pouvant pas utiliser leurs doigts, à 12 millions ceux ayant une vue déficiente, et à 80 millions les personnes présentant des troubles de l'audition.

La dernière décennie a vu un réel essor des technologies électroniques et informatiques miniaturisées, notamment avec la téléphonie cellulaire. Qu'en est-il réellement de l'accessibilité de ces technologies aux personnes présentant un handicap ?

Le projet dont il est question a pour but d'apporter un début de solution aux usagers avec besoins spéciaux, par le développement d'une plateforme cellulaire permettant l'ajout de périphériques adaptés et de fonctionnalités supplémentaires d'assistance.

Cette plateforme expérimentale comprend ainsi un module GSM permettant le dialogue avec le réseau cellulaire, un module GPS permettant la géolocalisation de l'utilisateur, ainsi qu'un certain nombre de dispositifs adaptés, tels qu'un écran à fort contraste, un clavier à grosses touches et aides lumineuses, etc...

Cette plateforme est prévue pour pouvoir recevoir d'autres fonctionnalités, tant matérielles (dans le but de l'adapter à davantage de personnes) que logicielles, afin de devenir un produit complet et répondant aux demandes des usagers à besoins spéciaux.

Mots Clés :
Téléphone cellulaire, usagers à besoins spéciaux, microcontrôleur, GSM, GPS

REMERCIEMENTS

Je tiens à remercier Monsieur Philippe Mabilleau, Directeur du Département de génie électrique et de génie informatique, et Professeur titulaire, pour son aide et son encadrement.

Un grand merci également à l'ensemble de l'équipe des techniciens du Département de génie électrique et de génie informatique, pour l'aide technique et parfois morale !

Il convient de souligner l'importance de l'appui financier de la Faculté de Génie ainsi que de la Fondation de l'Université.

N'oublions pas non plus le soutien moral de mon conjoint et de ma famille, qui a suivi l'avancement de ce projet à distance.

SOMMAIRE

1 Résumé ..... **ii**

2 Remerciements ..... **iii**

3 Sommaire ..... **iv**

4 Liste des Figures ..... **vi**

5 Liste des Tableaux ..... **viii**

6 Lexique ..... **x**

1 Introduction **1**

2 État de l'art et besoins **3**

2.1 Introduction 3

2.2 Analyse bibliographique 3

2.2.1 Article sur l'évaluation des besoins des usagers dans la téléphonie mobile 3

2.2.2 Article sur l'adaptation des technologies existantes pour les personnes avec des handicaps physiques 5

2.3 Besoins recensés 7

2.4 Populations visées 9

2.5 Technologies existantes 10

3 Plateforme matérielle et logicielle **11**

3.1 Introduction 11

3.2 Composants matériels 11

3.2.1 Schéma global 11

3.2.2 Module cellulaire GM862-GPS 12

3.2.3 Microcontrôleur LPC2106 13

3.2.4 Afficheur VFD2041 15

3.2.5 Clavier 16

3.2.6 Alimentation 17

3.3 Composants logiciels 18

3.3.1 OS temps réel 18

4 Mise en œuvre matérielle **20**

4.1 Introduction 20

4.2 Schémas électriques 20

4.2.1 Microcontrôleur LPC2106 20

4.2.2 Afficheur VFD2041 22

4.2.3 Clavier 22

4.2.4 Module Cellulaire GM862-GPS 32
4.2.5 Interfaces audio 33
4.2.6 Alimentation 38
4.3 Circuit imprimé 40
4.4 Versions futures 43

5 Mise en œuvre logicielle 44
5.1 Architecture 44
5.2 Couches de l'architecture logicielle 44
5.3 Développement des pilotes bas-niveau 45
5.3.1 Pilotes de l'UART0/1 (port série) 45
5.3.2 Pilotes du bus SPI 46
5.3.3 Pilotes du bus I²C 48
5.4 Développement des boîtes à outils intermédiaires 50
5.4.1 Boîte à outils relative au module cellulaire 50
5.4.2 Boîte à outils relative au clavier 54
5.4.3 Boîte à outils relative à l'afficheur 62
5.5 Développement de l'application haut niveau 65
5.5.1 Architecture 65
5.5.2 Cas d'utilisation 66
5.5.3 Composants 66
5.5.4 Fonctionnalités 69
5.5.5 Tests 69

6 Application validant la mise en œuvre 72
6.1 Retour sur les besoins 72
6.1.1 Sur le plan matériel 72
6.1.2 Sur le plan logiciel 73
6.2 Extensions possibles 73
6.2.1 Extensions matérielles 73
6.2.2 Extensions logicielles 75

7 Conclusion 79

A Annexes 81
A.1 Prototypes des fonctions principales 81
A.1.1 Pilotes bas-niveau 81
A.1.2 Boîtes à outils intermédiaires 83
A.1.3 Application 87
A.2 Code pertinent 89
A.2.1 Code de détection de l'appui d'une touche 89
A.2.2 Code de rafraîchissement des LEDs du clavier 91
A.2.3 Code d'allumage d'une LED (construction trame SPI) 92
A.3 Schéma électrique de l'ensemble du montage 93

B Bibliographie 98

LISTE DES FIGURES

Figure 3-1 – Schéma global des composants matériels 12
Figure 3-2 – Photos du module cellulaire GM862-GPS 13
Figure 3-3 – Photos du microcontrôleur LPC2106 15
Figure 3-4 – Photos de l'afficheur VFD2041 16
Figure 3-5 – Photos de l'afficheur VFD Active Matrix 16
Figure 3-6 – Clavier "*Button Pad 4x4*" 17
Figure 3-7 – Photos de la batterie Varta Lithium Polymer 3.7V 2200mAh 18
Figure 3-8 – Logo de µC/OS-II 19
Figure 4-1 – Schéma électrique du microcontrôleur LPC2106 21
Figure 4-2 – Schéma électrique de la connexion de l'afficheur VFD2041 22
Figure 4-3 – Schéma électrique des interfaces du clavier (touches seulement) 23
Figure 4-4 – Trames d'écriture SPI pour la détection d'une touche 24
Figure 4-5 – Trames de lecture SPI pour les touches, aucune touche appuyée 25
Figure 4-6 – Trames de lecture SPI pour les touches, touche 2 appuyée 25
Figure 4-7 – Schéma électrique du clavier 26
Figure 4-8 – Schéma électrique des interfaces du clavier (LEDs seulement) 27
Figure 4-9 – Trames d'écriture SPI pour les LEDs, aucune LED allumée 29
Figure 4-10 – Trames d'écriture SPI pour les LEDs, une LED allumée 30
Figure 4-11 – Schéma électrique du module cellulaire GM862-GPS 32
Figure 4-12 – Schéma global de l'interface audio 33
Figure 4-13 – Schéma électrique de l'entrée audio - microphone 34
Figure 4-14 – Schéma électrique de l'entrée audio - amplificateur 35
Figure 4-15 – Caractéristiques de l'entrée audio du module cellulaire 35
Figure 4-16 – Schéma électrique de la sortie audio 37
Figure 4-17 – Photo du convertisseur DC/DC 5V Vpack 38
Figure 4-18 –Répartition de la consommation en courant du circuit 39
Figure 4-19 – Schéma électrique de l'alimentation 40
Figure 4-20 – Photo du montage avant fabrication du PCB 40
Figure 4-21 – Schéma de l'empilement des composants sur le PCB 41

Figure 4-22 – Aspect final du PCB sous OrCAD Layout 42
Figure 4-23 – Photos du PCB avant et après assemblage des composants 42
Figure 5-1 – Organisation en couches de l'architecture logicielle 45
Figure 5-2 – Chronogramme d'utilisation du 74LS165 55
Figure 5-3 – Chronogramme d'utilisation des signaux de sélection pour le clavier 55
Figure 5-4 – Chronogrammes pour l'identification d'une touche 58
Figure 5-5 – Architecture globale de l'application 65
Figure 5-6 – Diagramme des cas d'utilisation de l'assistant mobile 66
Figure 5-7 – Diagramme d'activité « placer un appel » 70
Figure 5-8 – Diagramme d'activité « envoyer un SMS » 71
Figure 5-9 – Diagramme d'activité « obtenir la position GPS » 71
Figure 6-1 – Boutons d'un distributeur automatique de billets de banque 74
Figure 6-2 – Cas d'utilisation des services web de géolocalisation 75
Figure 6-3 – Architectures possibles pour des services de géolocalisation 75
Figure 6-4 – Cas d'utilisation pour connaitre le nom des rues avoisinantes 76
Figure 6-5 – Cas d'utilisation pour effectuer un appel d'urgence 77
Figure 6-6 – Cas d'utilisation pour connaitre les instructions pour parcourir un trajet 77
Figure A-1 – Schéma électrique global – Microcontrôleur et périphériques – 1 94
Figure A-2 – Schéma électrique global – Microcontrôleur et périphériques – 2 95
Figure A-3 – Schéma électrique global – GM862 et interface audio – 1 96
Figure A-4 – Schéma électrique global – GM862 et interface audio – 2 97

LISTE DES TABLEAUX

Tableau 2-1 – Barrières rencontrées dans l'utilisation des technologies sans-fil 4
Tableau 2-2 – Classement des fonctionnalités des cellulaires par ordre d'importance 5
Tableau 2-3 – Obstacles et fonctionnalités utiles dans l'utilisation d'un cellulaire 6
Tableau 4-1 – Utilisation des broches d'entrée-sortie du microcontrôleur 20
Tableau 4-2 – Organisation des trames SPI pour la détection de l'appui d'une touche 24
Tableau 4-3 – Logique dans le nommage des LEDs du clavier 27
Tableau 4-4 – Table de vérité, pour les anodes des LEDs 28
Tableau 4-5 – Table de vérité, pour les cathodes des LEDs 28
Tableau 4-6 – Table de vérité, l'allumage des LEDs 31
Tableau 4-7 – Calcul du gain à apporter par l'amplificateur d'entrée audio 36
Tableau 4-8 – Valeur des composants pour l'amplificateur d'entrée audio 36
Tableau 4-9 – Tensions et courants d'alimentation des principaux composants 39
Tableau 5-1 – Paramètres d'initialisation des UARTs 46
Tableau 5-2 – Paramètres d'initialisation du bus SPI 46
Tableau 5-3 – Configuration des broches d'entrée-sortie pour le bus SPI 47
Tableau 5-4 – Configuration du registre SPI Control Register 47
Tableau 5-5 – Paramètres d'initialisation du port I²C 48
Tableau 5-6 – Configuration des broches d'entrée-sortie pour le bus I²C 49
Tableau 5-7 – Configuration du registre I²C Control Set Register 49
Tableau 5-8 – Étapes d'initialisation de l'afficheur 50
Tableau 5-9 – Initialisation des connexions vers le module cellulaire 51
Tableau 5-10 – Initialisation des paramètres internes du module cellulaire 53
Tableau 5-11 – Étapes d'initialisation de l'afficheur 54
Tableau 5-12 – Initialisation de la structure des touches du clavier 56
Tableau 5-13 – Organisation des trames SPI pour la détection de l'appui d'une touche 57
Tableau 5-14 – Organisation des données pour la détection de l'appui d'une touche 57
Tableau 5-15 – MISO attendu en fonction de la colonne 58
Tableau 5-16 – Obtention de l'ID d'une touche à partir de ByteRec 59
Tableau 5-17 – Étapes d'initialisation de l'afficheur 62

Tableau 5-18 – Envoi de la commande « Move Cursor Back » à l'afficheur.....63
Tableau 5-19 – Construction du caractère spécial « é » pour l'afficheur.....64
Tableau 5-20 – Tâches créées pour l'application.....67
Tableau 5-21 – Boites aux lettres créées pour l'application.....68
Tableau 5-22 – Sémaphores créés pour l'application.....68

LEXIQUE

AAC	Augmentative and Alternative Communication
API	Application Programming Interface
AT	Assistive Technology
AT	ATtention, utilisé dans les Commandes Hayes pour modem
CR	Carriage Return
DC	Direct Current
GPIO	General Purpose Input Output
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile communications
I^2C	Inter-Integrated Circuit
ISR	Interrupt Service Routine
LED	Light-Emitting Diode
OS	Operating System
PCB	Printed Circuit Board
PLL	Phase-Locked Loop
SMS	Short Message Service
SPI	Serial Peripheral Interface
TVS	Transcient Voltage Suppressor
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
WAP	Wireless Access Protocol

CHAPITRE 1.

INTRODUCTION

Connaitre le nombre de personnes présentant un handicap est difficile. En effet, un handicap n'est pas juste un statut médical, mais plus une interaction entre ce statut et l'environnement.

Cependant, l'Organisation Mondiale pour la Santé estime à 600 millions le nombre de personnes invalides. L'Organisation des Nations Unies, elle, estime ce chiffre à 650 millions, soit presque 11% de la population mondiale. Aux États-Unis, cette population représente la 3ème minorité après les Hispaniques et Afro-américains. On l'estime à 32 millions d'adultes et 5 millions d'enfants et d'adolescents[1]. Le Bureau de Recensement des États-Unis, lui, estime à 53 millions la taille du marché de consommateurs présentant un handicap [SMITH-JACKSON et coll.,2003].

Devant de tels chiffres, il est nécessaire de s'intéresser aux adaptations faites pour ces personnes. Une « Technologie Assistive » ("*Assistive Technology*", ou AT) est un terme générique pour désigner des dispositifs et leurs modifications visant à rendre ces dispositifs accessibles aux personnes invalides. On peut citer les impressions en gros caractères, les claviers spéciaux, le Braille, la reconnaissance et la synthèse vocale, etc...

La fin du 20ème siècle et le début du 21ème siècle ont vu un réel essor des technologies électroniques et informatiques : micro-ordinateurs, Internet, téléphonie cellulaire, aide à la conduite en voiture... Les exemples sont très variés. Qu'en est-il réellement de l'accessibilité de ces technologies aux personnes présentant un handicap ?

Cette étude s'intéressera plus particulièrement au monde de la technologie cellulaire. En effet, c'est un domaine de première importance dans la mesure où il permet, en plus du

[1] Wikipedia – Disability – Demographics – http://en.wikipedia.org/wiki/Disability#Demographics

service rendu, une réelle amélioration des conditions de vie, une sécurité, une réduction de l'éloignement pour l'utilisateur et son entourage.

Pour ne citer que quelques chiffres, en Europe, on estime à 1 million le nombre de personnes ne pouvant pas utiliser leurs doigts, à 12 millions ceux ayant une vue déficiente, à 45 millions le nombre de personnes ne pouvant se déplacer sans aide extérieure et à 80 millions les personnes présentant des troubles de l'audition [GILL, 1996].

L'accès facilité à la technologie cellulaire présente donc un enjeu très important pour ces personnes : grand affichage présentant un bon contraste, clavier facilement utilisable, amplification du volume sonore, dispositifs de commande alternatifs (on pourrait suggérer la reconnaissance et la synthèse vocale [KOESTER, 2006]), etc...

Cependant, outre le service de base que l'on peut attendre de tout téléphone cellulaire, il peut être intéressant d'apporter un certain nombre de services additionnels tels que la localisation de l'utilisateur, l'aide à l'orientation, l'assistance en cas de problème. Le fait de placer des appels grâce à un téléphone accessible est déjà un premier pas. Mais en plus de l'accessibilité, ajouter un certain nombre de services d'assistance peut représenter une réelle plus-value pour la personne avec besoins spéciaux.

Sans vouloir apporter LA solution, cette étude se propose d'apporter un DÉBUT de solution par le développement d'une plateforme cellulaire permettant l'ajout de périphériques adaptés et de fonctionnalités supplémentaires d'assistance.

CHAPITRE 2.

ÉTAT DE L'ART ET BESOINS

2.1 Introduction

Avant de commencer un projet d'une telle envergure, il serait bon de prendre en compte les études qui ont déjà été réalisées sur le sujet. Ces études permettront de recenser quels sont, à l'heure actuelle, les besoins en technologie cellulaire pour les usagers à besoins spéciaux.

2.2 Analyse bibliographique

2.2.1 Article sur l'évaluation des besoins des usagers dans la téléphonie mobile

Quatre chercheurs ont réalisé une étude afin de regrouper un maximum de données à propos des barrières rencontrées par les personnes présentant des déficits cognitifs, physiques ou sensoriels sur l'utilisation des technologies de télécommunication. Cette étude présente également un ensemble de stratégies à suivre lors de la conception d'un appareil de télécommunication, afin d'en faciliter l'accès aux usagers à besoins spéciaux. Cette étude a été publiée sous la référence [MUELLER et coll., 2005].

Étude

Cette étude a amassé une quantité considérable d'informations sur l'utilisation des 4 types d'appareils suivants :

- Téléphones cellulaires ;
- Messages texte ;

- Appareils GPS ;
- Smartphones.

Cela s'est fait par le biais de sondages. On ne détaillera pas le mode opératoire, mais il est consultable dans l'article dont la référence figure ci-dessus.

Résultats

Ne figureront dans ce paragraphe que les résultats qui ont un intérêt direct avec ce projet. Voici pour commencer quelques statistiques (voir Tableau 2-1) issues de cette étude :

TABLEAU 2-1 – BARRIERES RENCONTREES DANS L'UTILISATION DES TECHNOLOGIES SANS-FIL

Barrière	Degré	Téléphones cellulaires	Messages texte	Appareils GPS	Smart phones
Difficultés rencontrées dans l'utilisation des produits sans-fil	Moyenne	26 %	13 %	19 %	15 %
	Grande	11 %	6 %	6 %	7 %
Limitations fonctionnelles causant cette difficulté	Dextérité	11 %	24 %	21 %	35 %
	Vision	22 %	37 %	39 %	81 %

Voici ensuite les aspects évalués concernant ce projet, et les remarques faites à ce sujet :

- Accès : toutes les personnes ayant participé à l'étude ont mentionné qu'elles souhaitaient mieux utiliser les technologies sans-fil. Parmi les suggestions, figurent la reconnaissance et la synthèse vocale, la dictée d'un texte, les ergots permettant de positionner les doigts, ainsi qu'une sortie en Braille (on pourrait utiliser un dispositif similaire au PAC Mate Portable Braille Display[1] dans ce projet) ;
- Fonctionnalités : les suggestions concernent un bouton à utiliser en cas d'urgence, la possibilité de changer la taille et la lisibilité de la police, la forme et la taille des boutons, la retranscription vocale des menus, la reconnaissance vocale, les ergots permettant de positionner les doigts, la personnalisation des sons, etc... ;
- Transportabilité / utilisabilité : les utilisateurs suggèrent l'ajout d'une poignée ou d'une attache derrière le téléphone permettant d'aider à le tenir. Un matériau antidérapant pourrait également être bénéfique ;
- Format : le téléphone doit être suffisamment grand pour qu'il soit facile à manipuler, mais suffisamment léger pour ne pas être pénible à porter ;
- Sécurité / respect de la vie privée : le téléphone doit permettre l'utilisation d'une oreillette afin de garder un minimum d'intimité.

[1] Freedom Scientific – http://www.freedomsci.com/products/fs/pacmate-braille-product-page.asp

Ces quelques suggestions pourront s'avérer utiles lors de la conception du téléphone concerné par ce document.

2.2.2 Article sur l'adaptation des technologies existantes pour les personnes avec des handicaps physiques

Cinq chercheurs ont réalisé une étude très similaire à la première, afin de savoir quelles étaient les fonctionnalités les plus demandées sur un téléphone cellulaire pour des usagers à besoin spéciaux, suivant leur degré de handicap. Cette étude s'avère très utile dans le cadre d'une étude des besoins dans la mesure où elle énumère les fonctionnalités les plus appréciées et les plus utiles pour ces usagers. Cette étude a été publiée sous la référence [NGUYEN et coll., 2006].

Étude

Cette étude porte sur 10 sujets, divisés en 3 groupes selon leur degré de handicap : « léger », « modéré » et « sévère ». En fonction des besoins de chacun, il est fourni un téléphone cellulaire cherchant à répondre à ces besoins. On évalue avant, pendant et après la performance et la satisfaction des participants quant à l'utilisation de ces appareils.

Les indicateurs de performance prenaient en compte les fonctionnalités suivantes :

- Appels vocaux (avec diverses méthodes) ;
- Messages textes et/ou courriel ;
- Répondeur ;
- Accès à des informations via le WAP ;
- Utilisation d'internet ;
- Autres fonctionnalités.

Résultats

Voici pour les 10 personnes les fonctionnalités classées par ordre d'importance :

TABLEAU 2-2 – CLASSEMENT DES FONCTIONNALITES DES CELLULAIRES PAR ORDRE D'IMPORTANCE

Fonctionnalité	Essentielle	Importante	Pratique	Non nécessaire
Haut-parleur	5	3	2	0
Composition vocale	5	3	2	0
Composition rapide	5	1	3	1

Répondeur	3	2	3	1
Écriture intuitive	3	1	2	4
Grand affichage	2	1	3	4
Accès via ordinateur	0	1	5	4
Bloc touches amovible	0	1	4	5
Clavier amovible	0	1	5	4
Écouteur sans fil	0	2	2	6
Écouteur avec fil	1	1	0	8

Le tableau ci-dessous Tableau 2-1 montre les obstacles rencontrés par les personnes de chaque groupe avant l'étude, ainsi que les fonctionnalités qui se sont avérées très importantes à la suite de l'étude.

TABLEAU 2-3 – OBSTACLES ET FONCTIONNALITES UTILES DANS L'UTILISATION D'UN CELLULAIRE

		Groupe 1	Groupe 2	Groupe 3
Degré		Léger	Modéré	Sévère
Obstacles avec un téléphone conventionnel		Touches et affichage trop petits Difficultés à lever ou tenir le téléphone	Ne peuvent utiliser complètement qu'un mode de communication (texte ou voix), ou les 2 mais partiellement	Utilisent une méthode de scan par interrupteur pour fonctionner avec un appareil AAC afin de communiquer Utilisation d'un téléphone impossible
Fonctionnalités	Composition vocale	100%	100%	100%
	Haut-parleur	100%	75%	100%
	Composition rapide	33%	50%	66%
	Répondeur	33%	75%	0%
	Écriture intuitive	33%	25%	33%

9 des 10 participants ont montré de bons, voire d'excellents résultats, tant en termes de performances qu'en termes de satisfaction.

Naturellement, ce sont les personnes du groupe 3 qui ont montré le plus gros progrès. En effet, auparavant, ces personnes ne pouvaient pas utiliser de téléphone sans l'aide d'un assistant du fait de leur handicap lourd. Les personnes des groupes 1 et 2 avaient pour la plupart déjà utilisé des téléphones mais pas au maximum de leurs capacités.

6 des 10 participants ont indiqué que le temps requis pour effectuer un certain nombre de manipulations essentielles a beaucoup diminué. 4 d'entre eux ont même vu le temps nécessaire pour passer un appel se réduire de quelques minutes à quelques secondes.
Quand 4 participants ne pouvaient pas répondre à un appel auparavant, tous ont pu le faire avec une technologie adaptée à leurs besoins.

Ce qui ressort principalement de cette étude sont les points suivants :

- Les personnes avec un handicap physique peuvent trouver une solution adéquate pour accéder aux équipements de télécommunication. Les appareils « mains libres » et les haut-parleurs se sont avérés essentiels ;
- Les personnes n'ayant jamais pu accéder à ce type de technologies, le peuvent maintenant lorsque leur équipement est adapté ;
- L'accès à la technologie cellulaire peut apporter un meilleur confort : accès, fonctionnalité, interactions sociales, indépendance, sécurité... ;
- L'apprentissage pour utiliser cette technologie s'avère déterminant.

Alors oui, les personnes présentant des besoins spéciaux peuvent se voir offrir des technologies adaptées répondant à leurs besoins.

2.3 Besoins recensés

Finalement, quels sont les besoins auxquels ce projet tentera de répondre ?

Il est souhaité, sur le plan matériel :

- une plateforme cellulaire de dimensions et poids raisonnables, pouvant se fixer sur un support quelconque, et n'ayant pas besoin d'être tenu près de l'oreille durant les communications, comme suggéré dans [NGUYEN et coll., 2001], table 4 ;
- un module cellulaire permettant de prendre en charge les communications avec le réseau cellulaire. Afin d'apporter un plus, il est souhaité que ce module permette une géolocalisation, à l'aide d'un GPS. L'utilité de la géolocalisation est notamment mise en évidence dans [KAASINEN, 2003] ;

> *" An efficient way of improving the usability of mobile services and applications is to adapt the contents and presentation of the service to each individual user and his/her current context of use. In this way, the amount of user interaction will be minimised: the user has quick access to the information or service that (s)he needs in his/her current context of use. The information can even be provided to the user automatically.*", [KAASINEN, 2003]

- un écran assez lumineux pour permettre une lecture relativement facile à des usagers présentant des troubles de la vision, comme également suggéré dans [NGUYEN et coll., 2001], table 4 ;
- un clavier dont les boutons sont suffisamment gros et espacés pour pouvoir être manipulé par des personnes présentant une dextérité réduite ;

> *"The size of the phone body was identified as important. Larger phones are easier to hold. Phones considered too thin or those having a small keypad (due to thinness) were not preferred by participants.*", [SMITH-JACKSON et coll.,2003]

- un haut-parleur permettant une sortie sonore avec un volume suffisamment fort, pour faciliter l'écoute des conversations aux usagers à l'ouïe affaiblie ;
- une batterie permettant d'avoir un système autonome sur le plan énergétique ;
- ... prévoir l'ajout d'autres périphériques matériels, tels que la possibilité de connecter des dispositifs d'adaptation spécifiques (bouton, commande par le souffle etc...).

Il est souhaité, sur le plan logiciel :

- pouvoir placer un appel, répondre à un appel et terminer une communication ;
- pouvoir envoyer un SMS d'urgence préprogrammé contenant la position GPS de l'usager, avec le choix d'un message parmi plusieurs ;
- ... prévoir des extensions à ce qui sera déjà implémenté, tel qu'un logiciel ouvert permettant l'intégration de nouvelles fonctionnalités exploitant les services de base déjà implantés: cellulaire, GPS, afficheur, clavier etc...

Par ailleurs, le produit ainsi conçu devra être simple d'utilisation, aussi bien sur le plan matériel que logiciel.

> *"We focused on novice users to gain information that would support design that was simple and easy to use, and that, consequently, could be used by more experienced individuals."*, [SMITH-JACKSON et coll.,2003]

Ainsi, il sera possible de proposer cet assistant mobile à un public plus vaste, incluant par exemple les aînés, qui bien souvent trouvent la technologie trop compliquée à aborder.

2.4 Populations visées

Comme ce projet n'en est qu'au début, c'est-à-dire que le but n'est pas de fournir un produit fini et industrialisable, il est pour le moment délicat d'évaluer précisément les populations visées. Cependant, il est possible d'en faire une première évaluation. L'assistant mobile sera destiné :

- Aux personnes mal- (par la taille et la luminosité de l'écran et des touches) ou non- (par le biais d'un afficheur Braille par exemple) voyantes ;
- Aux personnes malentendantes (par la possibilité de connecter un haut-parleur suffisamment puissant) ;
- Aux personnes présentant des problèmes de doigté, et pour lesquelles la manipulation de petits objets ou de petites touches est problématique (par la grosseur des touches du clavier et de l'ensemble du système) ;
- Aux personnes ayant une mobilité très restreinte, nécessitant l'emploi de contacteurs (dans la mâchoire, sur le sourcil, ...) par exemple, pour communiquer ;
- Aux personnes n'ayant pas de déficience particulière, mais souhaitant un produit simple d'utilisation ;
- De façon générale, à toute personne présentant un besoin particulier pour la téléphonie cellulaire, et pour laquelle ce besoin peut être assisté d'une manière électronique.

Cette liste est loin d'être exhaustive, et pourra être complétée lorsque le développement de cette plateforme sera poursuivi.

2.5 Technologies existantes

Il est délicat de recenser de façon exhaustive les technologies existantes, tant l'évolution de la recherche dans ce domaine est rapide. Lors des recherches effectuées au début de ce projet afin de déterminer les besoins, il était apparu que la plupart des expériences réalisées l'étaient à partir d'un téléphone cellulaire du commerce, modifié ensuite selon les besoins de l'usager. Ces recherches n'avaient pas permis d'identifier un appareil réalisé uniquement dans l'optique des usagers à besoins spéciaux.

Il existe des solutions proposées aux utilisateurs, comme les *Doro Matra HandleEasy 328gsm*[1] et *330gsm*[2], qui proposent un gros clavier, un écran de taille relativement importante, et une utilisation simple. Il est évident que compte tenu de la demande importante en téléphones cellulaires « adaptés », certains constructeurs travaillent sur la solution.

Cependant, le but de ce projet n'est pas de réaliser un simple téléphone cellulaire. Il est non seulement souhaité d'y ajouter des fonctionnalités telles que la géolocalisation par GPS, mais à terme, de prévoir la possibilité d'ajouter des périphériques propres à chaque type de handicap. Par exemple, connecter un clavier Braille à une personne non voyante, connecter un contacteur pour une personne ne pouvant communiquer qu'avec la mâchoire, etc...

---

[1] Doro Matra HandleEasy 328gsm – http://www.doro.fr/products/viewproduct?g=t&p=385

[2] Doro Matra HandleEasy 330gsm – http://www.doro.fr/products/viewproduct?g=t&p=398

CHAPITRE 3.

PLATEFORME MATERIELLE ET LOGICIELLE

3.1 Introduction

Au cours du chapitre précédent, il a été mis en évidence les différents besoins auxquels ce projet essayera de répondre. Dans la section suivante, les différents composants, matériels et logiciels, seront choisis afin de répondre à ces besoins.

3.2 Composants matériels

3.2.1 Schéma global

Voir Figure 3-1.

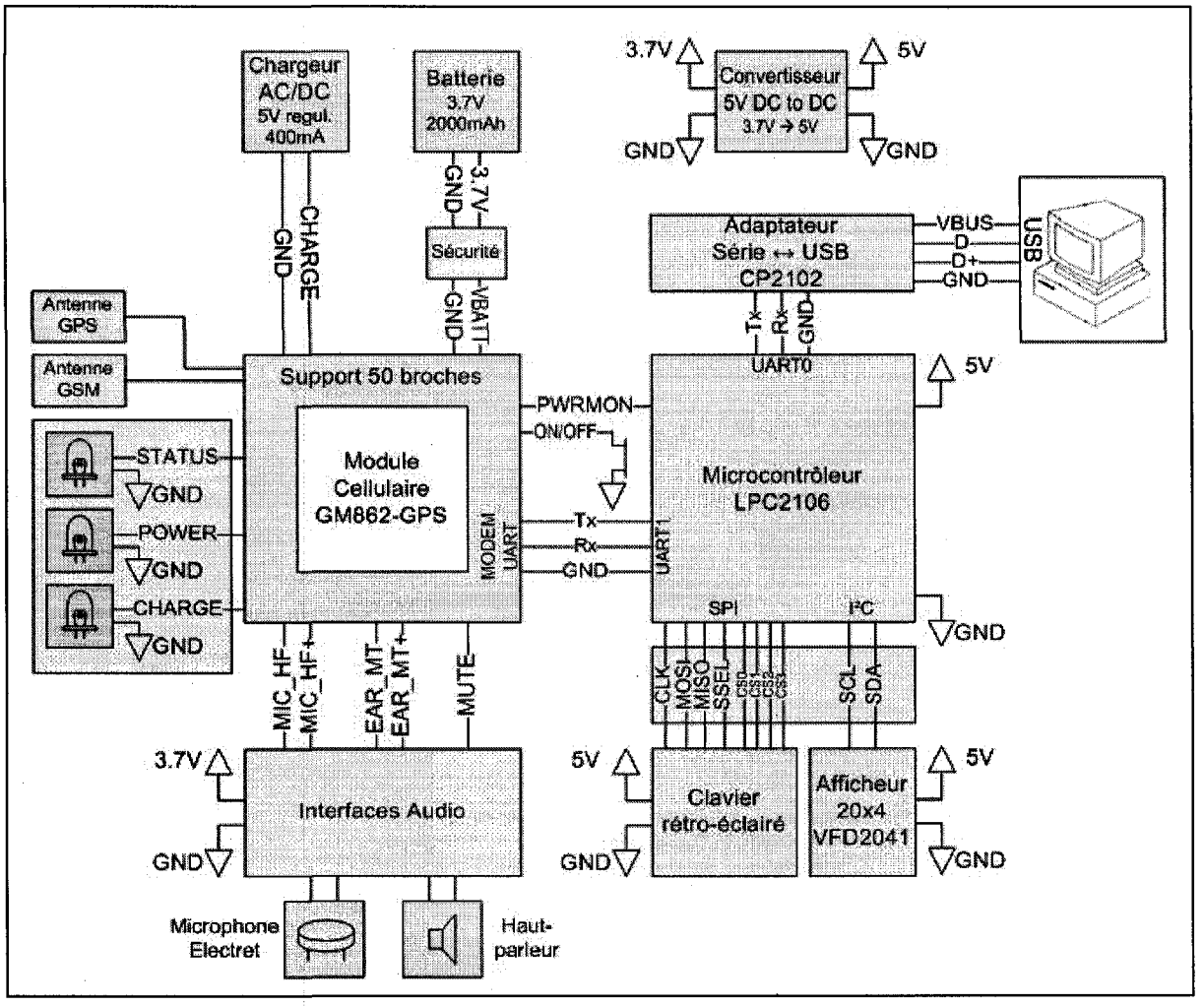


Figure 3-1 – Schéma global des composants matériels

3.2.2 Module cellulaire GM862-GPS

Par module cellulaire, on comprend les composants suivants :

- Module GSM GM862-GPS de Telit [TELIT, 2007hw], [TELIT, 2007sw] ;
- Antennes GSM et GPS ;
- Composants nécessaires à la réalisation du circuit audio.

Le module cellulaire est la base du projet. En effet, ce module consiste en une « boîte noire » sans autre interface que le port série, que l'on va exploiter afin d'avoir accès à toutes les fonctions de téléphonie et de géolocalisation dont on aura besoin.

Pour cela, on a choisi d'utiliser le module GM862-GPS de la compagnie Telit[1]. Ce module est un GSM quadri-bande, avec un récepteur GPS et un interpréteur Python. Voici une brève description du produit :

> *"The Telit GM862-GPS includes a 20 channels GPS receiver. It provides all the features of the GM862-QUAD version such as Voice, Circuit Switched Data transfer, Phonebook, SMS, four bands GSM capability, hot removal sensing on board SIM Reader, GPRS Class 10 and battery charger circuitry.*
> *Moreover, the GM862-GPS and GM862-QUAD-PY models, integrate the "EASY SCRIPT" functionality. This is a PYTHON engine script interpreter allowing self-controlled operations. With the EASY SCRIPT feature the GM862-GPS and GM862-QUAD-PY become a finite product, they just need your script to be run.*", [TELIT, 2008pd]

Le module n'a pas d'interface utilisateur directement utilisable. Cependant, on dispose des entrées-sorties nécessaires pour connecter directement l'électronique analogique des circuits audio, ainsi que de leur schéma électrique.

Bien que ce module dispose d'un interpréteur Python, le choix a été fait de l'utiliser en tant qu'esclave du microcontrôleur décrit ci-après, afin de faciliter l'ajout de périphériques. On communiquera donc avec ce module par port série, à l'aide notamment de commandes AT.

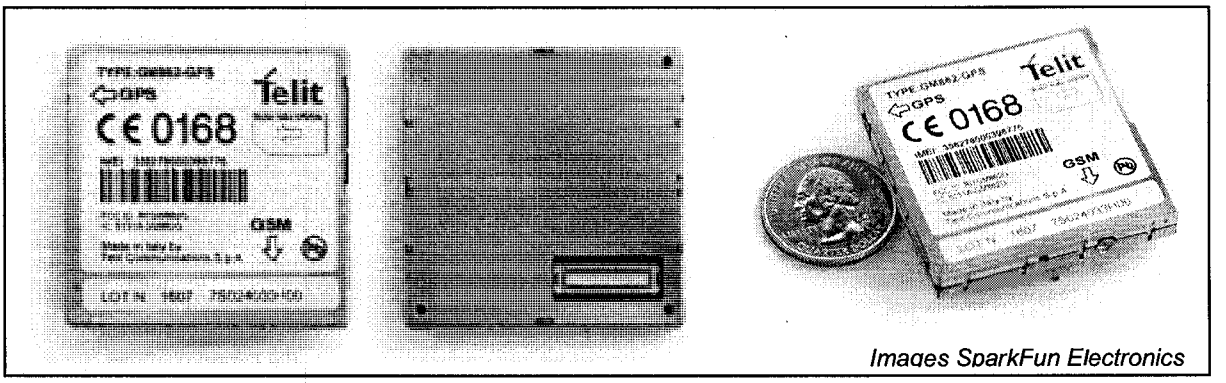


Figure 3-2 – Photos du module cellulaire GM862-GPS

3.2.3 Microcontrôleur LPC2106

Comme mentionné dans la section précédente, un microcontrôleur sera utilisé afin de pouvoir gérer de façon plus facile les périphériques interfacés. Ainsi, le microcontrôleur devient le composant central de la plateforme matérielle.

---

[1] Telit – http://www.telit.com

Pour cela, il a été choisi d'utiliser le microcontrôleur LPC2106 de NXP (Philips). Voici une brève description issue de sa *datasheet* :

> *"The LPC2104/2105/2106 are based on a 16/32-bit ARM7TDMI-S CPU with real-time emulation and embedded trace support, together with 128 kB of embedded high speed flash memory. A 128-bit wide memory interface and a unique accelerator architecture enable 32-bit code execution at maximum clock rate. For critical code size applications, the alternative 16-bit Thumb mode reduces code by more than 30 % with minimal performance penalty.*
>
> *Due to their tiny size and low power consumption, these microcontrollers are ideal for applications where miniaturization is a key requirement, such as access control and point-of-sale. With a wide range of serial communications interfaces and on-chip SRAM options up to 64 kB, they are very well suited for communication gateways and protocol converters, soft modems, voice recognition and low end imaging, providing both large buffer size and high processing power. Various 32-bit timers, PWM channels and 32 GPIO lines make these microcontrollers particularly suitable for industrial control and medical systems.*", [NXP, 2006ds]

Plus particulièrement :

> *"- 16/32-bit ARM7TDMI-S processor.*
> *- 16/32/64 kB on-chip static RAM.*
> *- 128 kB on-chip flash program memory. 128-bit-wide interface/accelerator enables high speed 60 MHz operation.*
> *- In-System Programming (ISP) and In-Application Programming (IAP) via on-chip boot loader software. Flash programming takes 1 ms per 512 B line. Single sector or full chip erase takes 400 ms.*
> *- Vectored Interrupt Controller with configurable priorities and vector addresses.*
> *- EmbeddedICE-RT interface enables breakpoints and watch points. Interrupt service routines can continue to execute whilst the foreground task is debugged with the on-chip RealMonitor software.*
> *- Embedded Trace Macrocell enables non-intrusive high speed real-time tracing of instruction execution.*
> *- Multiple serial interfaces including two UARTs (16C550), Fast I2C-bus (400 kbit/s) and SPI.*
> *- Two 32-bit timers (7 capture/compare channels), PWM unit (6 outputs), Real Time Clock and Watchdog.*
> *- Up to thirty-two 5 V tolerant general purpose I/O pins [...].*
> *- 60 MHz maximum CPU clock available from programmable on-chip Phase-Locked Loop with settling time of 100 ms.*
> *- On-chip crystal oscillator with an operating range of 1 MHz to 30 MHz.*
> *- Two low power modes, Idle and Power-down.*
> *- Processor wake-up from Power-down mode via external interrupt.*
> *- Individual enable/disable of peripheral functions for power optimization.*
> *- Dual power supply:*
> *- CPU operating voltage range of 1.65 V to 1.95 V (1.8 V ± 8.3 %).*

- I/O power supply range of 3.0 V to 3.6 V (3.3 V ± 10 %) with 5 V tolerant I/O pads.", [NXP, 2006ds]

Ainsi, on aura à disposition les ports UART0, UART1, SPI et I²C du LPC2106, ainsi que toutes ses entrées-sorties (GPIO). Dans l'optique de construire une plateforme ouverte permettant l'ajout futur de nombreux autres périphériques, la quantité de GPIO disponibles sera un atout certain.

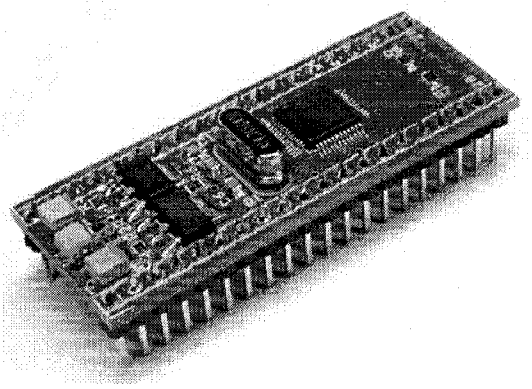


Image SparkFun Electronics

Figure 3-3 – Photos du microcontrôleur LPC2106

3.2.4 Afficheur VFD2041

L'écran que l'on souhaite utiliser doit privilégier la visibilité de son affichage à l'aspect esthétique. On cherche donc un écran avec une très forte luminosité. La technologie LCD, bien que très répandue dans de nombreuses applications, ne satisfait pas à cette condition.

La technologie PLEDs, proposée sur le site de Matrix Orbital[1] parait intéressante, mais les composants de base sont maintenant discontinus. La technologie VFD parait une très bonne alternative. Aussi, c'est le modèle VFD2041 du même fournisseur a été choisi.

Cet écran, de 20x4 caractères, permet également l'affichage de graphes ou d'images simples. Pilotable soit par port série (RS232), soit par I²C, c'est cette dernière solution qui a été retenue compte tenu de la disponibilité du port I²C sur le microcontrôleur LPC2106.

[1] Matrix Orbital – www.matrixorbital.com

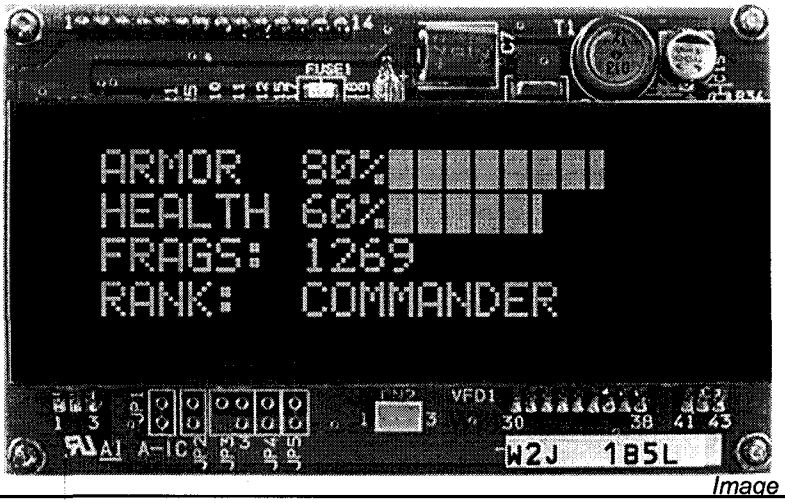


Image Matrix Orbital

Figure 3-4 – Photos de l'afficheur VFD2041

L'inconvénient majeur de cet afficheur est sa consommation électrique élevée, environ 250mA. Des recherches à la fin du projet de maîtrise ont permis de découvrir qu'il existe une technologie d'afficheurs encore plus lumineux et beaucoup plus économes en énergie, "*VFD Active Matrix*" de la compagnie Noritake[1] (la même compagnie qui fabrique l'écran VFD2041, sans l'interface I²C ajoutée par Matrix Orbital). L'utilisation de cette technologie peut être à prévoir dans des améliorations futures de la plateforme.

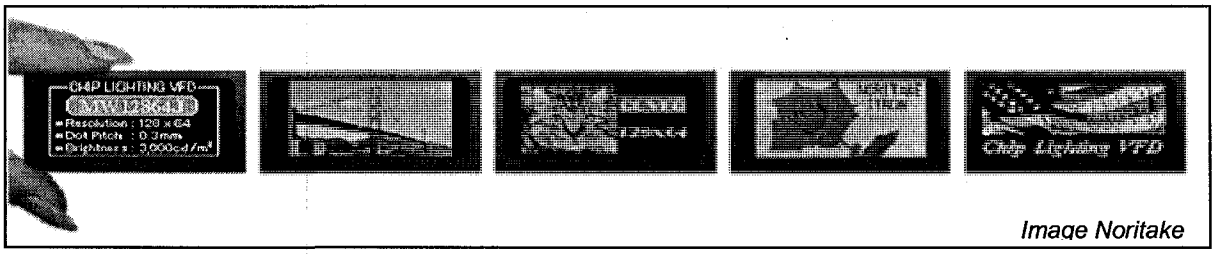


Image Noritake

Figure 3-5 – Photos de l'afficheur VFD Active Matrix

3.2.5 Clavier

La principale qualité cherchée pour le clavier de notre plateforme n'est pas la miniaturisation. Bien au contraire, un clavier aux touches larges et convenablement espacées, et pour lequel un appui touche est distinctement perceptible, sera utilisé. Un clavier de 4x4 touches permettra toutes les applications souhaitées.

---

[1] Noritake Electronics – http://www.noritake-elec.com

Pour cela, le clavier commercialisé par la société SparkFun Electronics[1] est intéressant. En plus de répondre aux exigences ci-dessus, ce modèle dispose en plus d'une LED tricolore par touche. On peut alors imaginer guider l'usager dans l'utilisation du téléphone en utilisant un code de couleurs (vert pour « OK », rouge pour « Corriger »...).

L'ajout de symboles sur chaque touche est réalisé grâce à l'impression de ces symboles sur des étiquettes transparentes vendues dans le commerce. Chaque touche étant équipée d'une LED tricolore, le nombre de fils à connecter est très élevé. Afin de n'utiliser qu'un minimum de GPIO du microcontrôleur, le choix a été fait d'utiliser quelques registres à décalages, transistors et autres composants afin d'interfacer le clavier et ses LEDs. Ceci sera détaillé à la section 4.2.3.

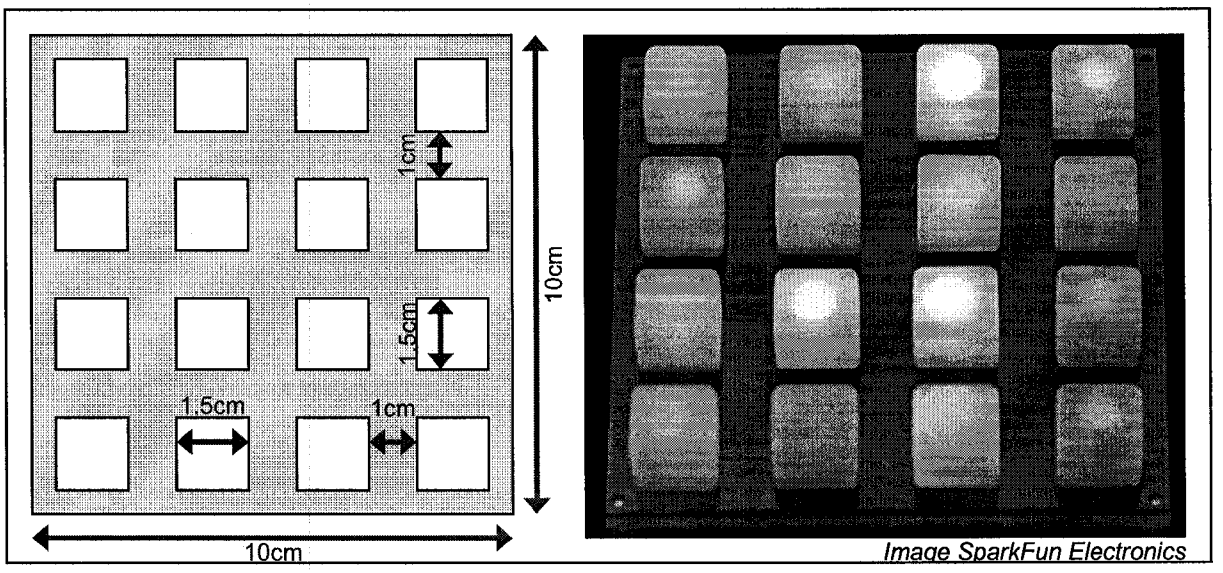


Figure 3-6 – Clavier "*Button Pad 4x4*"

3.2.6 Alimentation

La conception de l'alimentation du système est de toute première importance puisque c'est elle qui donnera à l'ensemble de la plateforme un caractère « portable ». En effet, il est difficilement concevable d'avoir un téléphone cellulaire nécessitant d'être en permanence relié par un fil à une alimentation continue fixe.

[1] Sparkfun Electronics – http://www.sparkfun.com

Le module cellulaire étant doté d'un chargeur de batterie en interne, ce dernier sera utilisé. Il faut donc une batterie Lithium-Ion 3.7V. La capacité de la batterie devra être assez élevée compte tenu du nombre de composants à alimenter, et surtout, compte tenu de la consommation électrique de l'afficheur. Le choix se porte sur une batterie Varta Lithium Polymère, 3.7V et de capacité 2200mAh, ainsi qu'une alimentation régulée 5V 400mA pour recharger cette batterie.

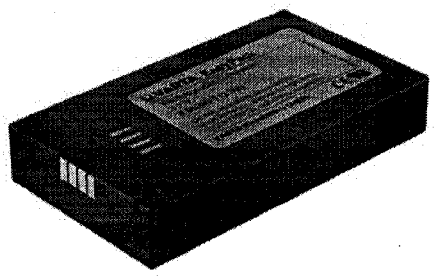


Image Newark

Figure 3-7 – Photos de la batterie Varta Lithium Polymer 3.7V 2200mAh

Les détails concernant la mise en œuvre de cette batterie, ainsi que la consommation en courant de chaque composant feront l'objet de la section 4.2.6.

3.3 Composants logiciels

Si les composants matériels ont été choisis en premier de par leur pertinence vis-à-vis des spécificités des usagers, il convient d'établir une stratégie en termes de logiciel.

3.3.1 OS temps réel

Les 2 possibilités offertes étaient d'avoir une application régie AVEC ou SANS OS. Après un premier développement des drivers, il a finalement été décidé d'utiliser les fonctionnalités de l'OS temps-réel µC/OS-II.

Voici une brève introduction à µC/OS-II[1]:

> *"µC/OS-II, The Real-Time Kernel is a highly portable, ROMable, very scalable, preemptive real-time, multitasking kernel (RTOS) for microprocessors and microcontrollers. µC/OS-II can manage up to 255 tasks and provides the following services:*
> *- Semaphores*
> *- Mutual Exclusion Semaphores (to reduce priority inversions)*
> *- Event Flags*
> *- Message Mailboxes*
> *- Message Queues*
> *- Task Management (Create, Delete, Change Priority, Suspend / Resume etc.)*
> *- Fixed Sized Memory Block management*
> *- Time Management*
> *- Timer Management*
> *µC/OS-II runs on a large number of processor architectures."*

L'utilisation de cet OS permettra ainsi une gestion aisée des périphériques, grâce à la possibilité de créer un certain nombre de taches. L'architecture définie pour l'application finale sera détaillée dans la section 5.1.

Figure 3-8 – Logo de µC/OS-II

[1] Micrium – µC/OS-II Kernel Overview – http://micrium.com/products/rtos/kernel/rtos.html

CHAPITRE 4.

MISE EN ŒUVRE MATERIELLE

4.1 Introduction

Une fois les composants matériels choisis, intervient l'interfaçage de ces divers composants. Il convient de préciser que la mise en œuvre matérielle n'a pas été de directement produire un circuit imprimé à partir des schémas électriques. En effet, l'assemblage de tous les composants matériel a été progressif, partant dans un premier temps du microcontrôleur et de l'écran, puis les ajouts successifs du clavier et du téléphone, sur *breadboard* (plaque à trous de prototypage rapide).

4.2 Schémas électriques

4.2.1 Microcontrôleur LPC2106

Connexions

Utilisation des broches d'entrées-sorties :

TABLEAU 4-1 – UTILISATION DES BROCHES D'ENTREE-SORTIE DU MICROCONTROLEUR

Broche	Fonction	Signal	Connexion
P0.0	UART0	Tx0 – Transmission	Ordinateur (via CP2102)
P0.1		Rx0 – Réception	
P0.2	I²C	SCL – Serial CLock	Afficheur VFD2041
P0.3		SDA – Serial DAta	
P0.4	SPI	SCK – Serial ClocK	Clavier (touches et LEDs)
P0.5		MISO – Master In Slave Out	
P0.6		MOSI – Master Out Slave In	
P0.7		SSEL – Slave SELect	

P0.8	UART1	Tx1 – Transmission	Module cellulaire GM862-GPS
P0.9		Rx1 – Réception	
P0.22	Chip Select Clavier	CS0 – Chip Select 74LS164	Clavier (touches et LEDs)
P0.23		CS1 – Chip Select 74HC4094	
P0.24		CS2 – Chip Select 74HC4094	
P0.25		CS3 – Chip Select 74LS165	
P0.27	GM862	PWRMON – Surveille alim. cellulaire	Module cellulaire GM862-GPS

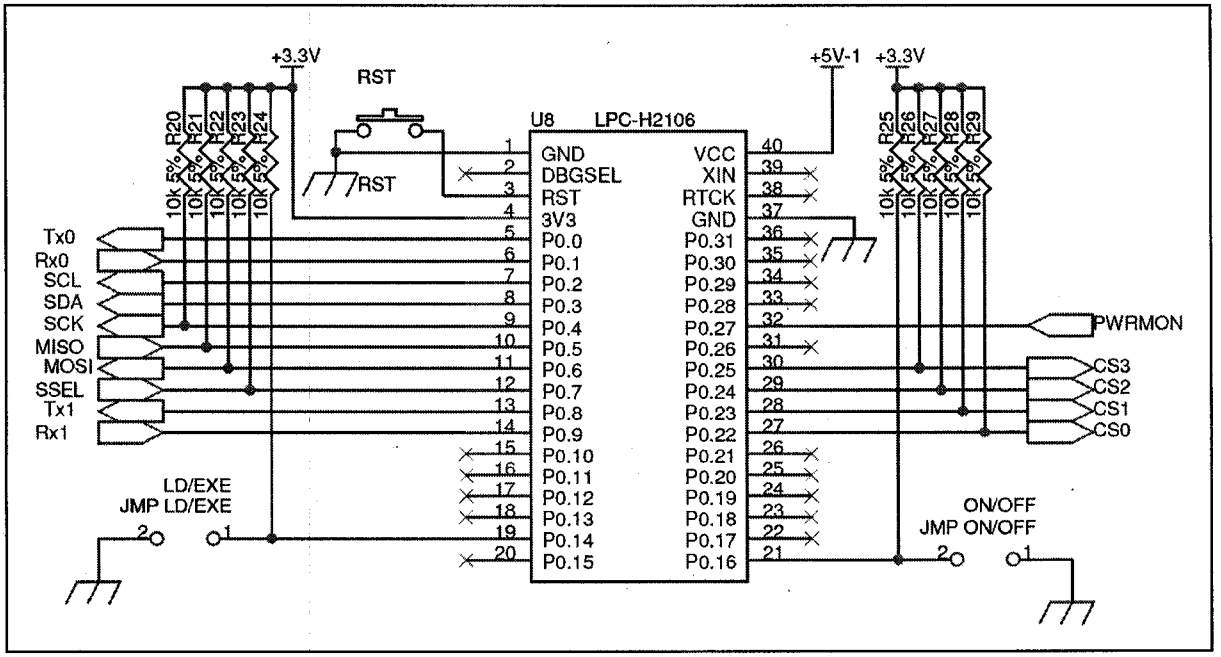


Figure 4-1 – Schéma électrique du microcontrôleur LPC2106

Les détails sur l'utilisation des signaux ci-dessus seront abordés dans les sections correspondant aux périphériques concernés.

Tests

Afin de tester l'alimentation et le bon fonctionnement du microcontrôleur, il est possible de charger une simple application faisant clignoter une LED que l'on branche en série avec une résistance sur une broche d'entrée-sortie libre.

4.2.2 Afficheur VFD2041

Connexion

L'afficheur peut être piloté soit par port série, soit par I²C. Dans la mesure où il est préférable de conserver la disponibilité des 2 ports UART du microcontrôleur LPC2106 pour la programmation et le module cellulaire, c'est l'I²C qui a été choisi.

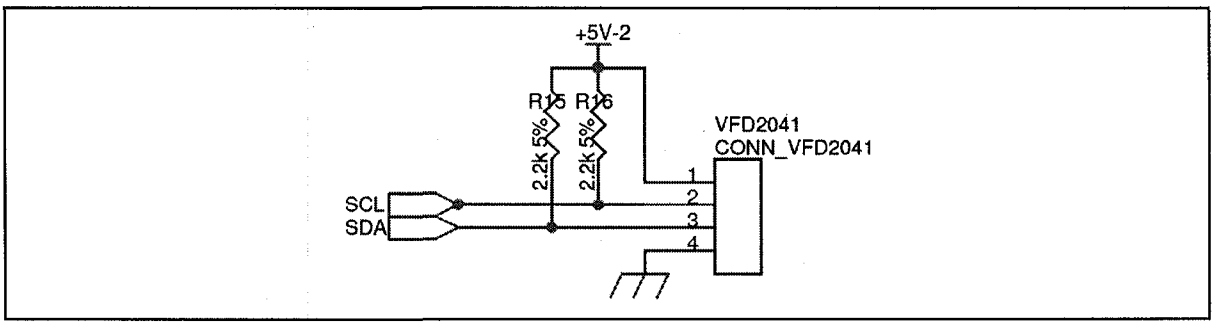


Figure 4-2 – Schéma électrique de la connexion de l'afficheur VFD2041

Le connecteur utilisé pour l'afficheur est un connecteur de type Molex 4 broches femelle. On utilise 2 résistances de *pull-up* afin de respecter les spécifications mentionnées dans le guide d'utilisateur du LPC2106 [NXP, 2003um] à la page 112 figure 17 : "*I²C Bus Configuration*".

Tests

L'afficheur est livré avec une application permettant, une fois installée sur l'ordinateur, de le commander via un port COM de l'ordinateur. Cela permet de tester les fonctions de base.

4.2.3 Clavier

Il aurait été simple de connecter directement les lignes et les colonnes des touches du clavier aux GPIOs du microcontrôleur. Cependant, le clavier possédant une LED tricolore sous chaque touche, l'ensemble (touches et LEDs) aurait nécessité 24 broches du microcontrôleur.

Le but de ce projet étant de concevoir une plateforme de base, permettant par la suite l'ajout de périphériques multiples, il est dans notre intérêt de minimiser au maximum le nombre de

broches utilisées du microcontrôleur, quitte à complexifier l'électronique d'interface. C'est le choix qui a été réalisé.

Interfaçage des touches

Pour interfacer les touches du clavier, on a donc choisi des registres à décalages fonctionnant par SPI. Ainsi, le microcontrôleur pourra, grâce au 74LS164, écrire sur les lignes du clavier, et grâce au 74LS165, lire sur les colonnes. Dans le but d'obtenir les signaux souhaités, on utilise un inverseur 74HC04 comme précisé dans le schéma Figure 4-3 ci-dessous. Il est conseillé au lecteur de consulter les *datasheets* des composants ci-nommés.

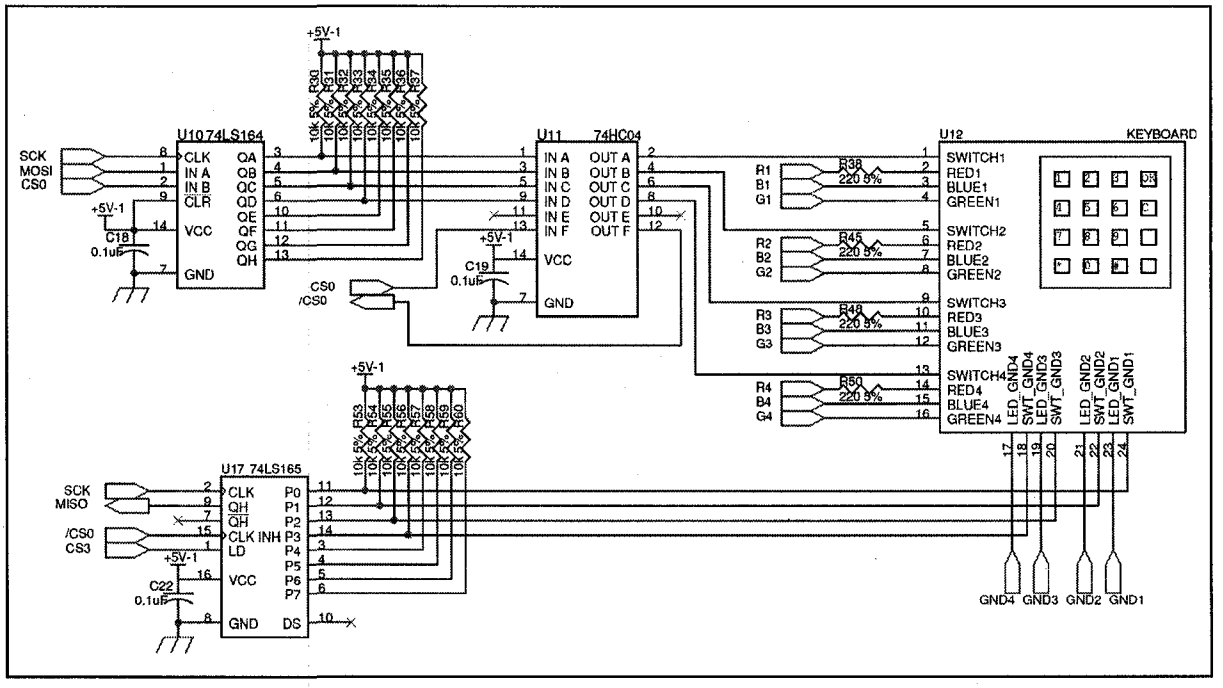


Figure 4-3 – Schéma électrique des interfaces du clavier (touches seulement)

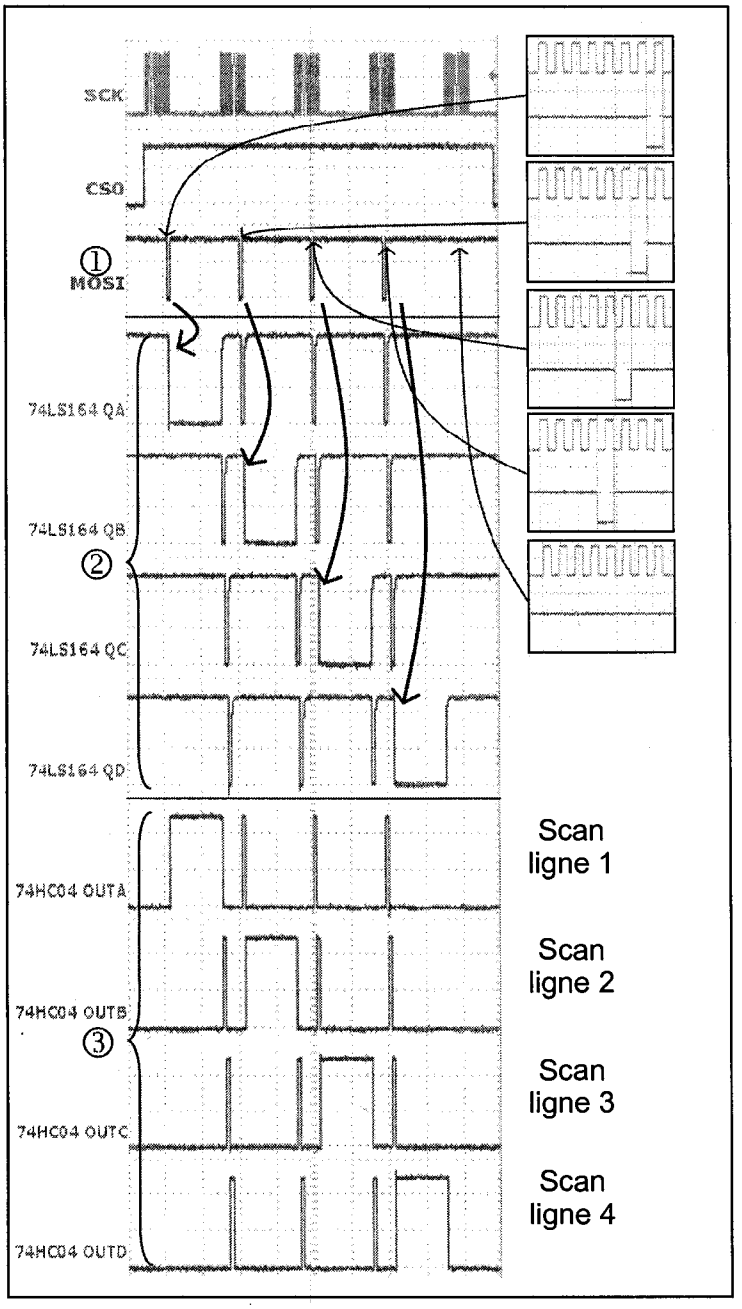
Pour détecter une touche pressée, on réalise un balayage de la matrice de contacts formée par le clavier. Pour cela, on vient écrire des « 1 » successivement sur chaque ligne « i » (les autres lignes étant à « 0 »). La détection d'un « 1 » sur une colonne « j » viendra indiquer que la touche (ligne i, colonne j) a été pressée (les autres colonnes étant à « 0 »).

La détection de l'appui d'une touche se fait en 5 étapes, soit 5 cycles d'écriture-lecture sur le port SPI. Pour un $n^{ème}$ cycle donné, on écrit sur le port SPI à l'aide du signal MOSI ("*Master Out Slave In*"). On lira ainsi l'état du clavier au $(n+1)^{ème}$ cycle SPI.

TABLEAU 4-2 – ORGANISATION DES TRAMES SPI POUR LA DETECTION DE L'APPUI D'UNE TOUCHE

Cycle (SCK)	Trame 1	Trame 2	Trame 3	Trame 4	Trame 5
Écriture (MOSI)	Ligne 1	Ligne 2	Ligne 3	Ligne 4	~~(à Ignorer)~~
Lecture (MISO)	~~(à Ignorer)~~	Ligne 1	Ligne 2	Ligne 3	Ligne 4

Les trames d'écriture sont les suivantes :



① Les 5 trames MOSI envoyées par le microcontrôleur présentent 1 des 8 bits à 0, à chaque fois décalé d'un front d'horloge.

② Puis, le 74LS164 convertit ces trames MOSI en 8 signaux parallèles (seuls 4 sont utilisés).

③ Comme on souhaite pouvoir écrire des « 1 » successivement sur chaque ligne pour pouvoir détecter ou non un appui touche, on utilise l'inverseur 74HC04.

Figure 4-4 – Trames d'écriture SPI pour la détection d'une touche

Si aucune touche n'est appuyée :

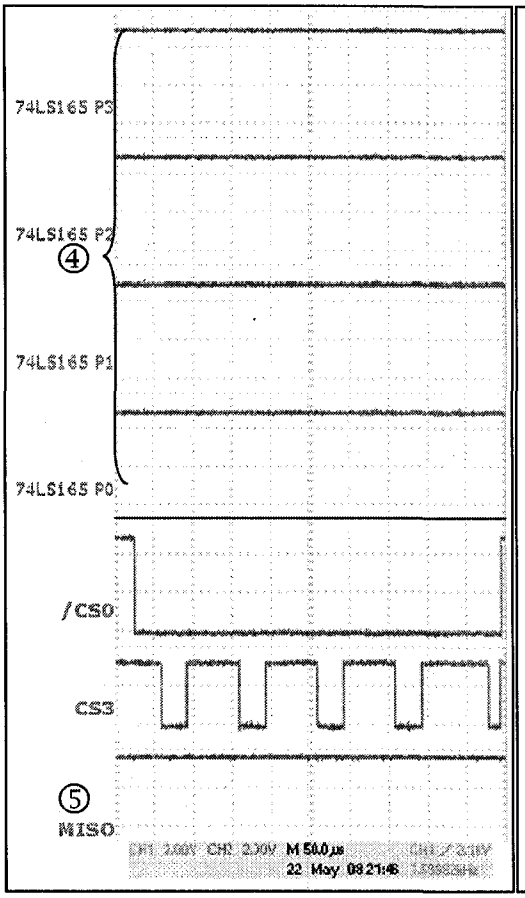


Figure 4-5 – Trames de lecture SPI pour les touches, aucune touche appuyée

Si la touche 2 est appuyée :

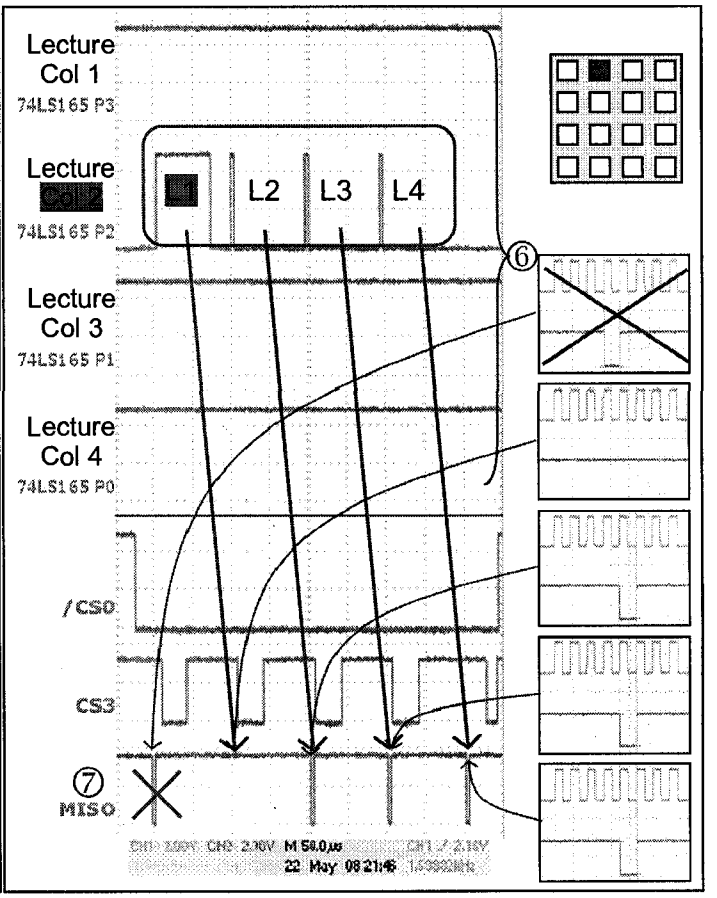


Figure 4-6 – Trames de lecture SPI pour les touches, touche 2 appuyée

Si aucune touche n'est appuyée :

④ Bien que l'on ait appliqué des « 1 » successivement à chaque ligne, aucune touche n'est appuyée donc les colonnes sont à 5V grâce aux résistances de *pull-up* en aval.

⑤ MISO est donc à « 1 » en permanence, et aucun appui de touche n'est détecté.

Si la touche 2 est appuyée :

⑥ Une touche appuyée revient à établir un contact entre la ligne et la colonne correspondantes (ligne 1, colonne 2, dans notre cas). Ainsi, sur la broche permettant de lire la colonne 2, on retrouve ce qui a été écrit dans la ligne 1 en ③.

⑦ Le composant 74LS165 sert alors à lire sur les colonnes les signaux et en forme une trame appelée MISO, qui sera lue lors du prochain échange écriture-lecture SPI.

L'interfaçage du clavier sur le port SPI est ainsi fait. Les détails concernant la reconnaissance des touches à partir de la trame MISO seront abordés dans la section « Mise en œuvre logicielle ».

Interfaçage des LEDs

Pour les LEDs, un principe similaire à l'interfaçage des touches est utilisé. En effet, des registres à décalage permettent d'allumer ou éteindre indépendamment chacune des LEDs à partir d'une trame MOSI provenant du microcontrôleur.

Le clavier est composé de LEDs tricolores (R, B, G), mais on pilote chaque couleur indépendamment. On considèrera donc que chaque touche a 3 LEDs distinctes : une rouge, une bleue et une verte.

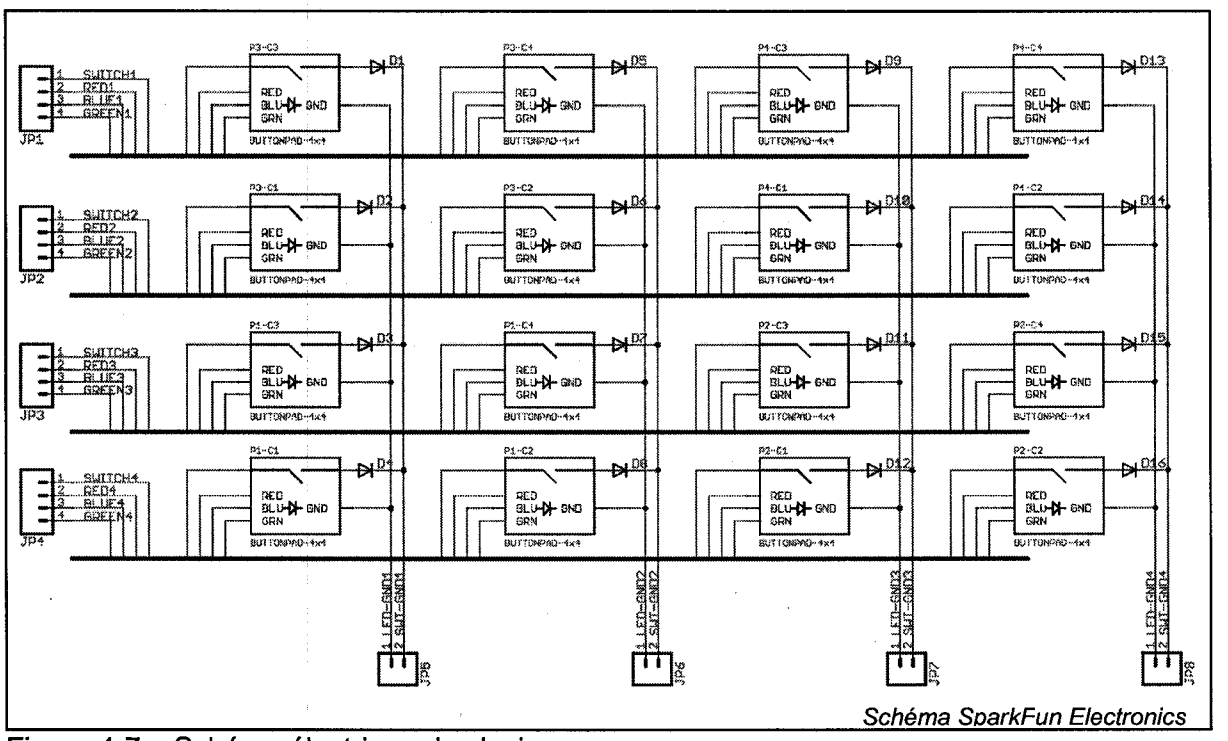


Figure 4-7 – Schéma électrique du clavier

L'idée, c'est que pour allumer une LED, il faut y appliquer une différence de potentiel. En d'autres termes, mettre +5V sur l'anode, c'est-à-dire la ligne du clavier, et 0V sur la cathode, c'est-à-dire la colonne correspondante.

A l'inverse, pour éteindre (c'est-à-dire ne pas allumer) une LED, il suffit d'y appliquer 0V à l'anode et 5V à la cathode.

Voici la logique utilisée pour nommer les LEDs :

TABLEAU 4-3 – LOGIQUE DANS LE NOMMAGE DES LEDS DU CLAVIER

	Colonne 1	Colonne 2	Colonne 3	Colonne 4
Ligne 1	R1 GND4 B1 GND4 G1 GND4	R1 GND3 B1 GND3 G1 GND3	R1 GND2 B1 GND2 G1 GND2	R1 GND1 B1 GND1 G1 GND1
Ligne 2	R2 GND4 B2 GND4 G2 GND4	R2 GND3 B2 GND3 G2 GND3	R2 GND2 B2 GND2 G2 GND2	R2 GND1 B2 GND1 G2 GND1
Ligne 3	R3 GND4 B3 GND4 G3 GND4	R3 GND3 B3 GND3 G3 GND3	R3 GND2 B3 GND2 G3 GND2	R3 GND1 B3 GND1 G3 GND1
Ligne 4	R4 GND4 B4 GND4 G4 GND4	R4 GND3 B4 GND3 G4 GND3	R4 GND2 B4 GND2 G4 GND2	R4 GND1 B4 GND1 G4 GND1

Voici le schéma électrique dont le fonctionnement est détaillé en Figure 4-8 ci-dessous :

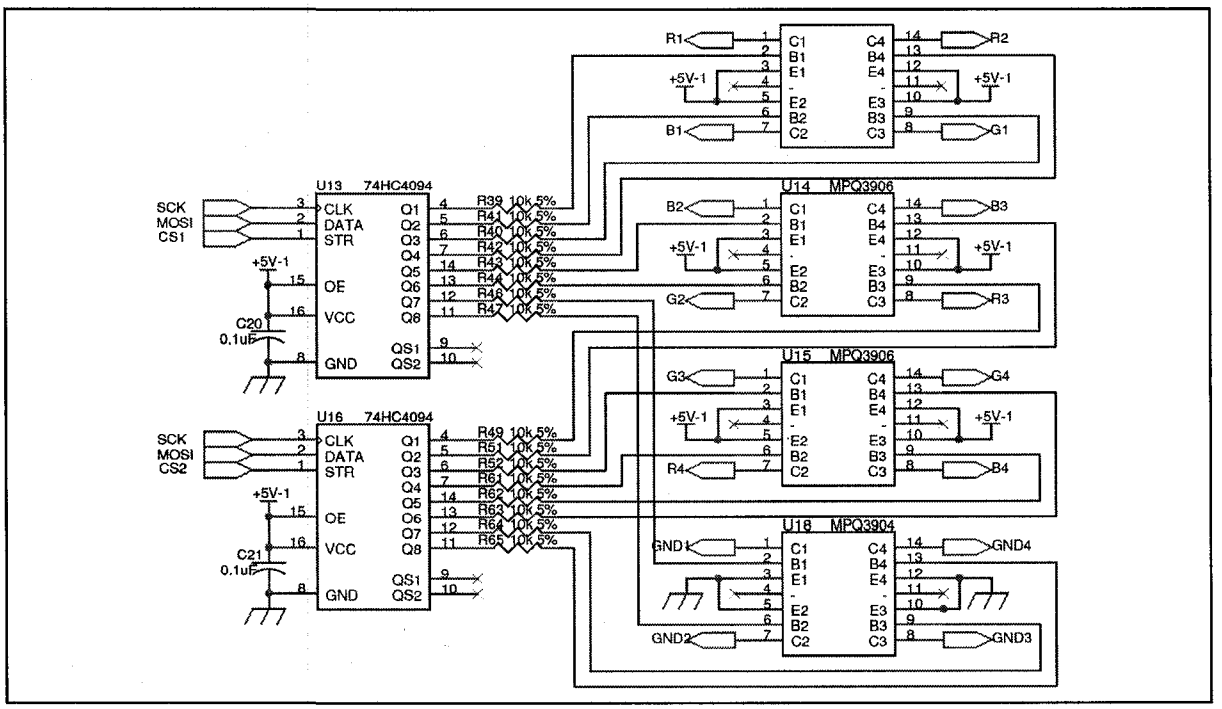


Figure 4-8 – Schéma électrique des interfaces du clavier (LEDs seulement)

A partir de la trame MOSI provenant du microcontrôleur, dont la construction sera détaillée dans la section 5.4.2, le 74HC4094 met à 1 ou à 0 les sorties correspondant aux LEDs concernées.

Les 3 MPQ3906 sont des blocs de 4 transistors PNP, dont les bases sont reliées aux sorties du 74HC4094, et dont les collecteurs sont connectés aux anodes des LEDs (lignes du

clavier). Le MPQ3904 est un bloc de 4 transistors NPN, dont les bases sont reliées aux sorties des 74HC4094, et dont les collecteurs sont connectés aux cathodes des LEDs (colonnes du clavier).

Table de vérité, pour les anodes des LEDs :

TABLEAU 4-4 – TABLE DE VERITE, POUR LES ANODES DES LEDS

SI	MOSI au $n^{ème}$ front d'horloge (n=1..6)	1	0
ALORS	74HC4094 à la sortie Q_n (n=1..6)	1	0
	Base du MPQ3906 correspondante	1	0
	Collecteur du MPQ3906 correspondant	0	1
	Anode de la LED correspondante	0	1
	Si la cathode associée est à 0	-	LED allumée
	Si la cathode associée est à 1	LED éteinte	-

Table de vérité, pour les cathodes des LEDs :

TABLEAU 4-5 – TABLE DE VERITE, POUR LES CATHODES DES LEDS

SI	MOSI au $n^{ème}$ front d'horloge (n=7..8)	1	0
ALORS	74HC4094 à la sortie Q_n (n=7..8)	1	0
	Base du MPQ3904 correspondante	1	0
	Collecteur du MPQ3904 correspondant	1	0
	Cathode de la LED correspondante	1	0
	Si l'anode associée est à 0	LED éteinte	-
	Si l'anode associée est à 1	-	LED allumée

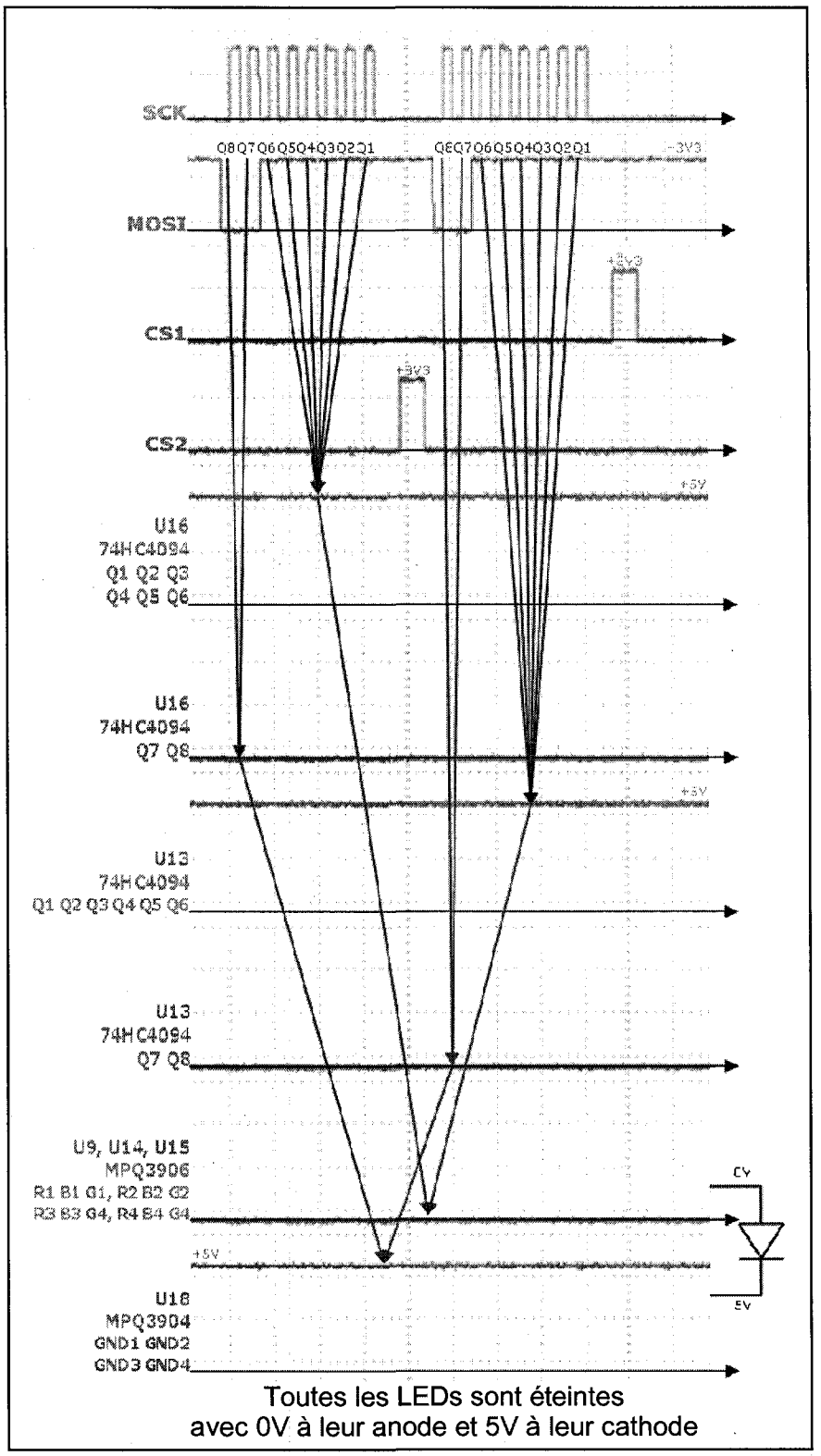
En résumé :

- Pour allumer une LED, il faut que le bit MOSI correspondant à son anode soit à 1 et que le bit MOSI correspondant à sa cathode soit à 0 ;
- Pour éteindre une LED, il faut que le bit MOSI correspondant à son anode soit à 0 et que le bit MOSI correspondant à sa cathode soit à 1.

Par exemple, pour allumer la LED bleue de la 1ère ligne et de la 2ème colonne du clavier, ce qui correspond à B1 GND3, il faut que le bit MOSI correspondant à B1 soit à 0 et le bit MOSI correspondant à GND3 soit à 1.

Les graphes en Figure 4-9 et Figure 4-10 permettent de visualiser les trames correspondantes.

Le 1er graphe est lorsque toutes les LEDs sont éteintes :



Toutes les LEDs sont éteintes
avec 0V à leur anode et 5V à leur cathode

On vérifie ainsi que les bits de MOSI correspondant aux cathodes (Q7 et Q8) sont effectivement à 0,

et que les bits de MOSI correspondant aux anodes (Q1..Q6) sont effectivement à 1.

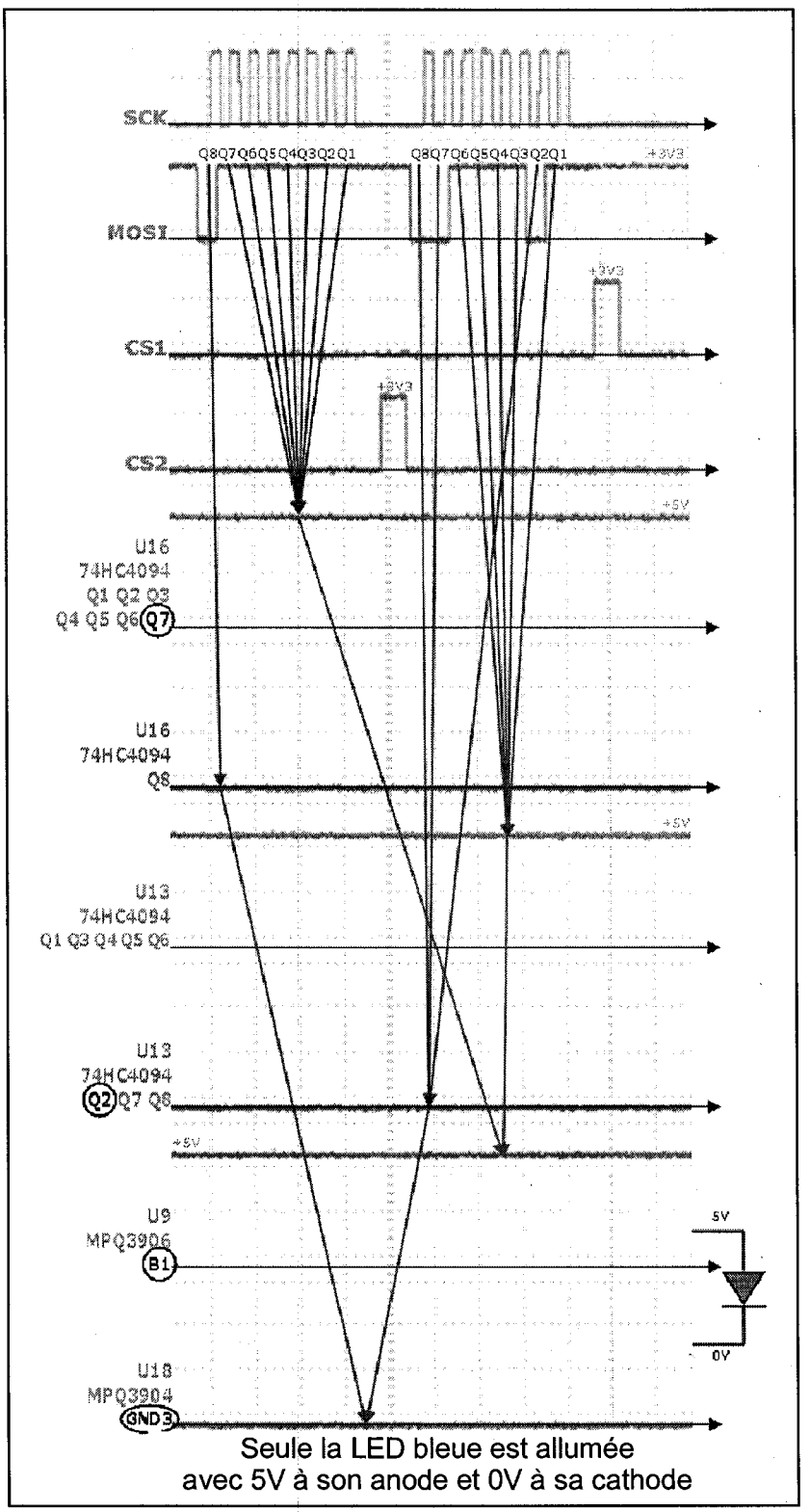
Ainsi, les collecteurs des transistors PNP (MPQ3906), reliés aux anodes, sont à 0,

et les collecteurs des transistors NPN (MPQ3904), reliées aux cathodes, sont à 1.

Aucune LED ne peut donc être allumée.

Figure 4-9 – Trames d'écriture SPI pour les LEDs, aucune LED allumée

Le 2ème graphe est lorsque la LED bleue de la 1ère ligne et 2ème colonne (B1 GND3) est allumée :



On vérifie ainsi que les bits de MOSI correspondant aux cathodes AUTRES QUE GND3 sont effectivement à 0, celui correspondant à GND3 étant à 1,

et que les bits de MOSI correspondant aux anodes AUTRES QUE B1 sont effectivement à 1, celui correspondant à B1 étant à 0.

Ainsi, les collecteurs des transistors PNP (MPQ3906), reliés aux anodes AUTRES QUE B1, sont à 0, celui correspondant à B1 étant à 1,

et les collecteurs des transistors NPN (MPQ3904), reliées aux cathodes AUTRES QUE GND3, sont à 1, celui correspondant à GND3 étant à 0.

Seule la LED bleue de la 1ère ligne et de la 2ème colonne peut donc être allumée.

C'est ainsi que l'on pourra commander chaque couleur de chaque LED tricolore de façon indépendante, et ce en n'utilisant que les 4 signaux du port SPI et 4 signaux de contrôle CS0... CS3.

Figure 4-10 – Trames d'écriture SPI pour les LEDs, une LED allumée

Pour résumer les trames à envoyer :

TABLEAU 4-6 – TABLE DE VERITE, L'ALLUMAGE DES LEDS

ID Touche	Couleur	74HC4094 (U16)									74HC4094 (U13)								
		Q8	Q7	Q6	Q5	Q4	Q3	Q2	Q1	MOSI	Q8	Q7	Q6	Q5	Q4	Q3	Q2	Q1	MOSI
		GND4	GND3	G4	B4	R4	G3	B3	R3		GND2	GND1	G2	B2	R2	G1	B1	R1	
1	R	**1**	0	1	1	1	1	1	1	BF	0	0	1	1	1	1	1	**0**	3E
	B															1	**0**	1	3D
	G															**0**	1	1	3B
2	R	0	**1**	1	1	1	1	1	1	7F	0	0	1	1	1	1	1	**0**	3E
	B															1	**0**	1	3D
	G															**0**	1	1	3B
3	R	0	0	1	1	1	1	1	1	3F	**1**	0	1	1	1	1	1	**0**	BE
	B															1	**0**	1	BD
	G															**0**	1	1	BB
4	R	0	0	1	1	1	1	1	1	3F	0	**1**	1	1	1	1	1	**0**	7E
	B															1	**0**	1	7D
	G															**0**	1	1	7B
5	R	**1**	0	1	1	1	1	1	1	BF	0	0	1	1	**0**				37
	B												1	**0**	1	1	1	1	2F
	G												**0**	1	1				1F
6	R	0	**1**	1	1	1	1	1	1	7F	0	0	1	1	**0**				37
	B												1	**0**	1	1	1	1	2F
	G												**0**	1	1				1F
7	R	0	0	1	1	1	1	1	1	3F	**1**	0	1	1	**0**				B7
	B												1	**0**	1	1	1	1	AF
	G												**0**	1	1				9F
8	R	0	0	1	1	1	1	1	1	3F	0	**1**	1	1	**0**				77
	B												1	**0**	1	1	1	1	6F
	G												**0**	1	1				5F
9	R	**1**	0	1	1	1	1	1	**0**	BE	0	0	1	1	1	1	1	1	3F
	B						1	**0**	1	BD									
	G						**0**	1	1	BB									
10	R	0	**1**	1	1	1	1	1	**0**	7E	0	0	1	1	1	1	1	1	3F
	B						1	**0**	1	7D									
	G						**0**	1	1	7B									
11	R	0	0	1	1	1	1	1	**0**	3E	**1**	0	1	1	1	1	1	1	BF
	B						1	**0**	1	3D									
	G						**0**	1	1	3B									
12	R	0	0	1	1	1	1	1	**0**	3E	0	**1**	1	1	1	1	1	1	7F
	B						1	**0**	1	3D									
	G						**0**	1	1	3B									
13	R	**1**	0	1	1	**0**				B7	0	0	1	1	1	1	1	1	3F
	B			1	**0**	1	1	1	1	AF									
	G			**0**	1	1				9F									
14	R	0	**1**	1	1	**0**				77	0	0	1	1	1	1	1	1	3F
	B			1	**0**	1	1	1	1	6F									
	G			**0**	1	1				5F									
15	R	0	0	1	1	**0**				37	**1**	0	1	1	1	1	1	1	BF
	B			1	**0**	1	1	1	1	2F									
	G			**0**	1	1				1F									
16	R	0	0	1	1	**0**				37	0	**1**	1	1	1	1	1	1	7F
	B			1	**0**	1	1	1	1	2F									
	G			**0**	1	1				1F									

Tests

Le clavier sera testé dès que les pilotes bas-niveau et la boîte à outils lui correspondant seront développés, voir la section 5.4.2.

4.2.4 Module Cellulaire GM862-GPS

Connexion

Le module cellulaire est connecté au microcontrôleur et à ses interfaces de la façon suivante :

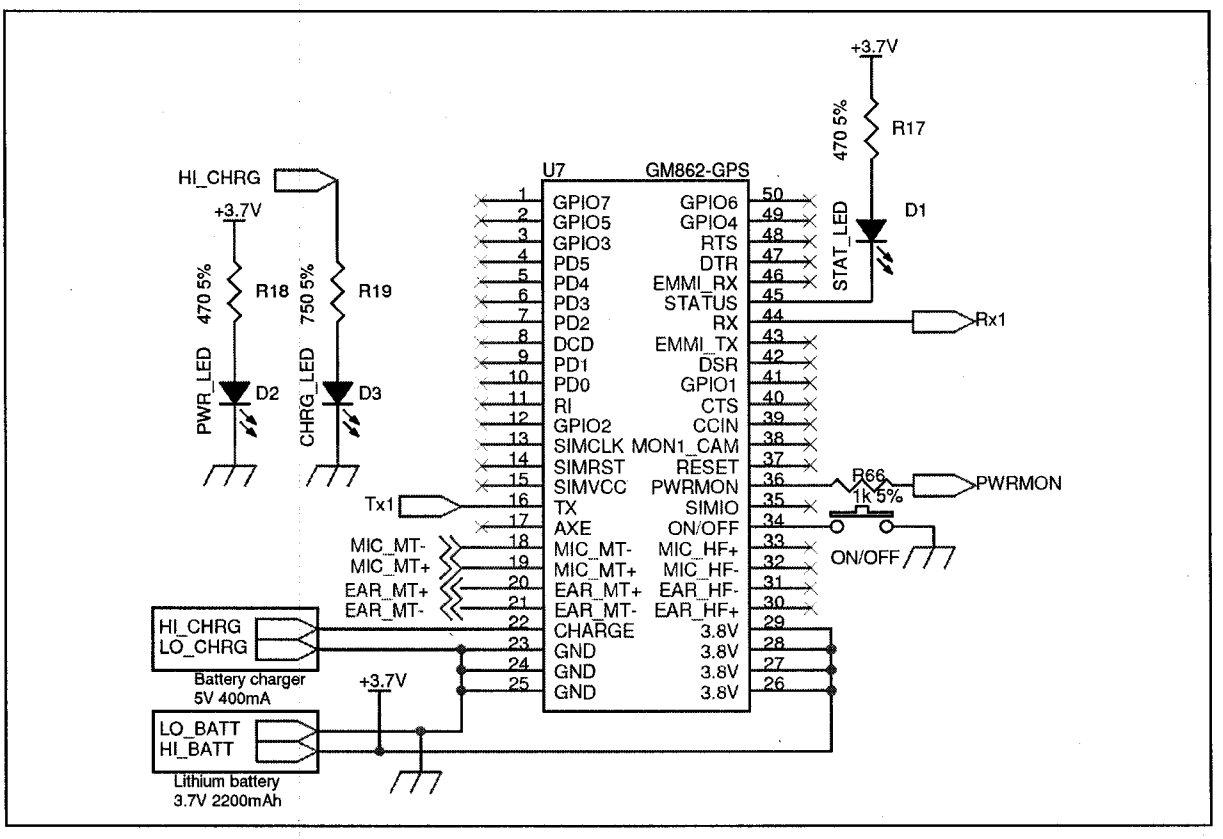


Figure 4-11 – Schéma électrique du module cellulaire GM862-GPS

4.2.5 Interfaces audio

Le module cellulaire étant un module dépourvu de toute interface directement utilisable (c'est-à-dire sans clavier, écran, microphone, haut-parleur) par l'utilisateur, il convient de prévoir les interfaces audio.

Pour cela, il est possible d'exploiter les schémas prévus dans le document [TELIT, 2007hw], aux sections "*Audio Section Overview*" et "*Output Lines (Speaker)*".

Vue d'ensemble du système audio

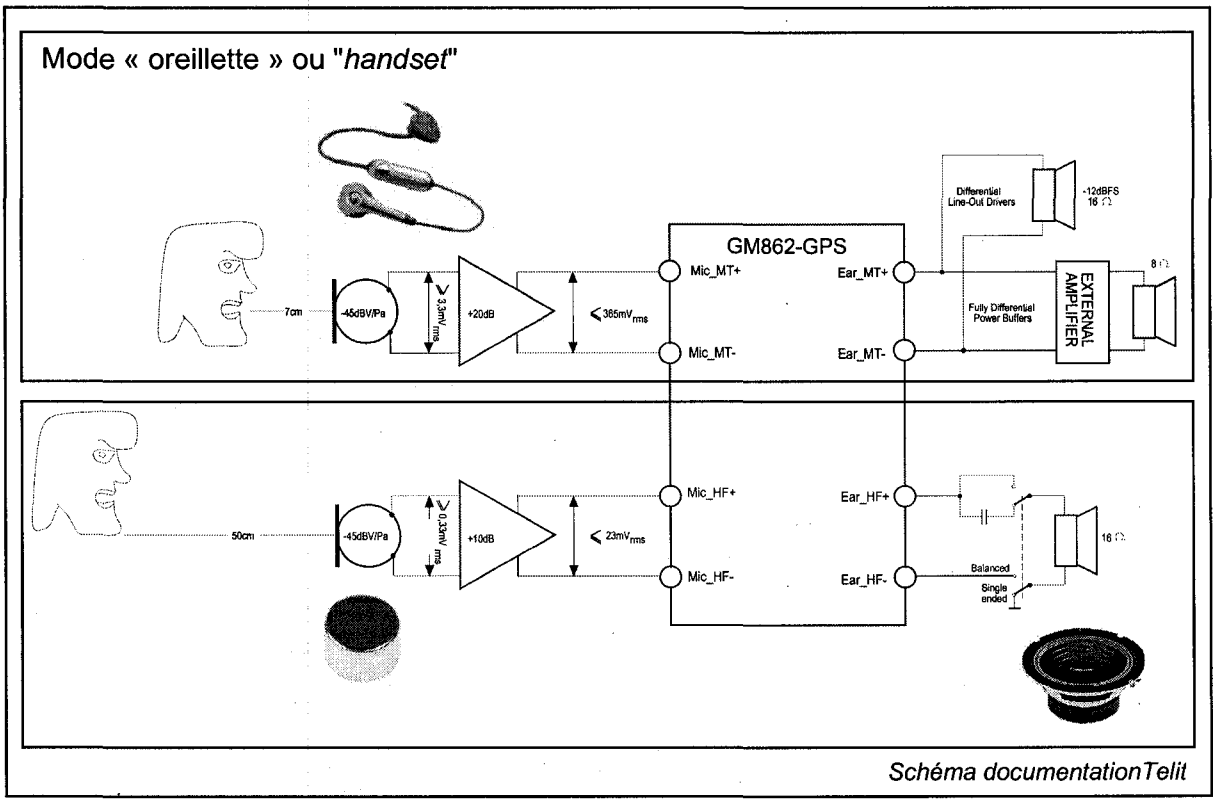


Figure 4-12 – Schéma global de l'interface audio

Modes d'utilisation

Comme on peut le voir sur la Figure 4-12 de la page précédente, il existe 2 modes d'utilisation principaux.

Le mode « oreillette » ou "*handset*", permet une utilisation en mode privé du téléphone, c'est-à-dire que la communication se fait par le biais d'un écouteur équipé d'un petit microphone

Le mode « mains libres » ou "*hands free*", permet une utilisation commune ou à distance comme par exemple dans une voiture. La communication se fait par le biais d'un microphone électret omnidirectionnel, et d'un haut-parleur relativement puissant. Il a été choisi de privilégier le 2ème mode d'utilisation, le mode « mains libres » ou "*hands free*".

Entrée audio - Microphone

Ce montage est destiné aux entrées MIC_HF+ et - du module GM862-GPS, par le biais d'un amplificateur (voir page suivante). Cependant, pour un meilleur fonctionnement, ce montage a été connecté aux entrées MIC_MT+ et – du module cellulaire. Cela n'en altère pas le bon fonctionnement.

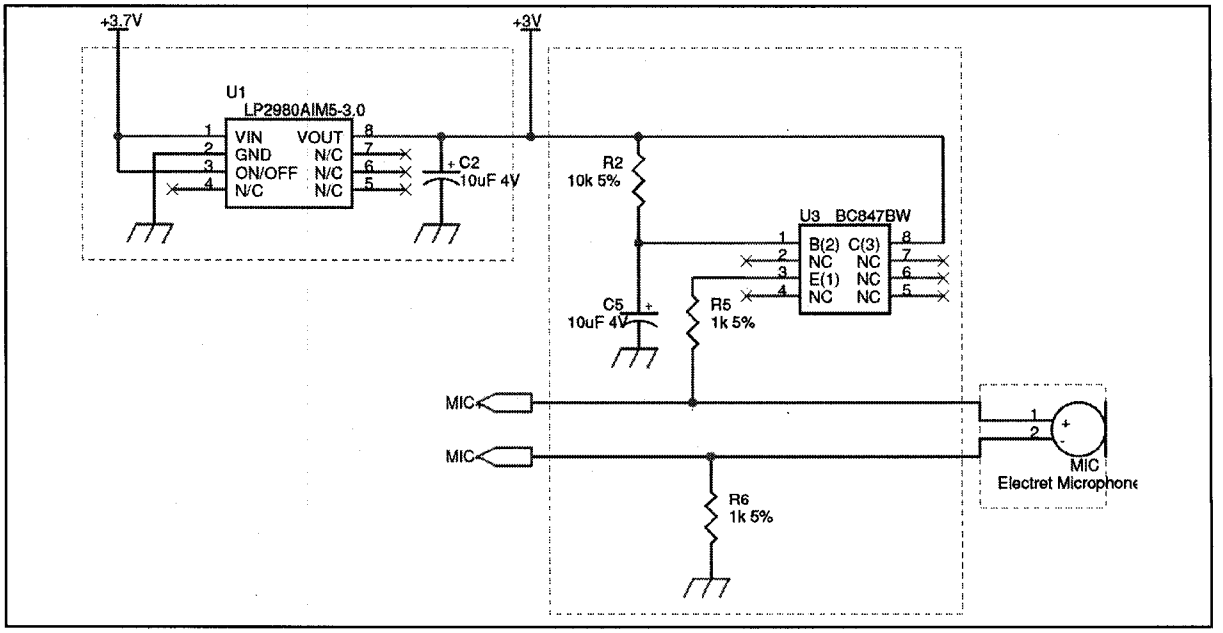


Figure 4-13 – Schéma électrique de l'entrée audio - microphone

Le microphone doit avoir une sensibilité avoisinant -45dBVRMS/Pa. Or, celui qui est disponible a une sensibilité de -69dBVRMS/Pa, pouvant fournir en sortie une tension de 10-69/20 = 0.35mV. Il faudra donc compenser par un gain plus important sur l'étage suivant.

Les modifications qui sont à apporter sont sur le gain de l'amplificateur, voir section suivante.

Entrée audio - Amplificateur

Ce montage a à son entrée un montage avec microphone de type électret (voir page précédente).

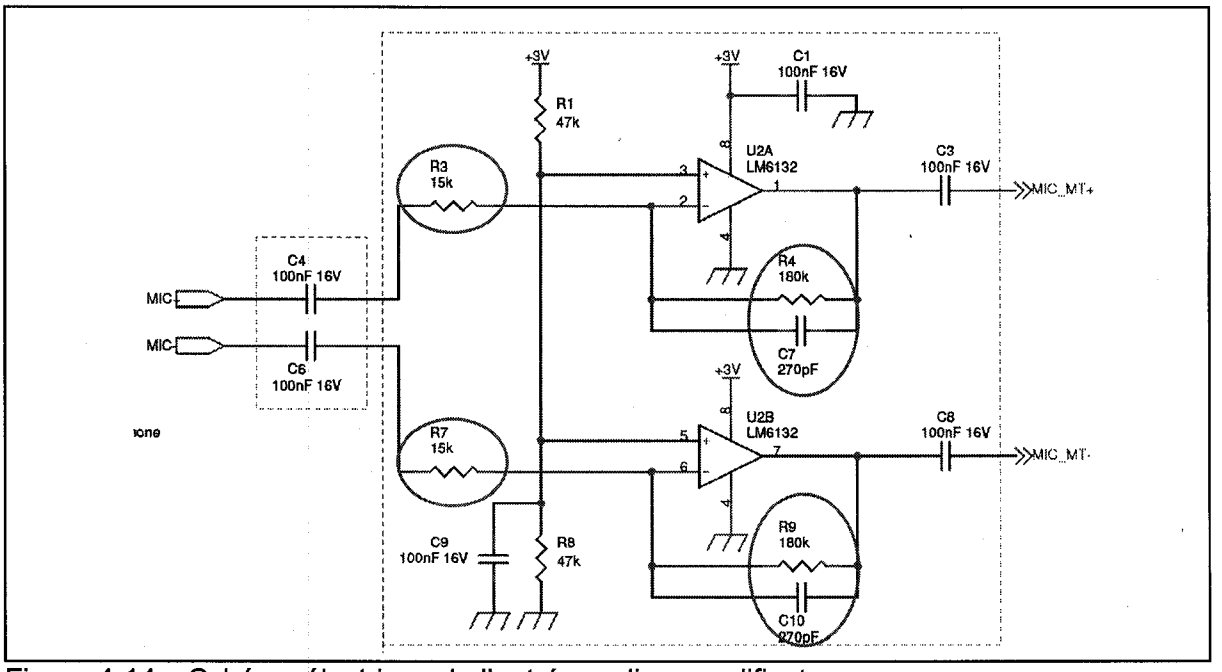


Figure 4-14 – Schéma électrique de l'entrée audio - amplificateur

Voici les caractéristiques d'entrée du module GM862-GPS :

"Mic_HF" 2nd differential microphone path

• **line coupling**	**AC**
• **line type**	**balanced**
• **coupling capacitor**	**≥ 100nF**
• **differential input resistance**	**50kΩ**
• **differential input voltage**	**≤ 65mV$_{pp}$ (23mV$_{rms}$)**
• **microphone nominal sensitivity**	**-45 dBV$_{rms}$/Pa**
• **analog gain suggested**	**+10dB**
• **echo canceller type**	**car kit hands-free**

DocumentationTelit

Figure 4-15 – Caractéristiques de l'entrée audio du module cellulaire

L'entrée MIC_HF supporte donc une tension différentielle de 65mVpp, soit -32.77dB.

Voici le calcul du gain à apporter par l'amplificateur en environnement non bruyant :

TABLEAU 4-7 – CALCUL DU GAIN A APPORTER PAR L'AMPLIFICATEUR D'ENTREE AUDIO

	Sensibilité du microphone	-69 dB
+	Atténuation due à l'écart entre l'utilisateur et le microphone	-4.7 dB
+	Atténuation supplémentaire en mode « mains libres »	-10 dB
+	Gain du microphone en environnement non bruyant	+20 dB
+	Gain de l'amplificateur	**GA**
=	Tension différentielle d'entrée du module	-32.77dB

On arrive alors à la somme :

$$(-69\ dB) + (-4.7\ dB) + (-10\ dB) + (20\ dB) + GA = (-32.77\ dB) \quad (4\text{-}1)$$

Soit :

$$GA = (-32.77\ dB) - (-69\ dB) - (-4.7\ dB) - (-10\ dB) - (20\ dB) \quad (4\text{-}2)$$

$$GA = (-32.77 + 69 + 4.7 + 10 - 20)\ dB \quad (4\text{-}3)$$

$$GA \approx +30.93\ dB \quad (4\text{-}4)$$

Ainsi, le gain à apporter par l'amplificateur est, :

$$gain = 10^{\frac{30,93}{20}} = 35,2 \quad (4\text{-}5)$$

Calcul des composants à modifier :

Pour ajuster le gain, il faut jouer sur les valeurs des composants R3, R4, R7, R9, C7 et C10 tel que vu à la page 51 du document [TELIT, 2007hw] :

$$gain = \frac{R_4}{R_3} = \frac{R_9}{R_7} \text{ et } C_7 = C_{10} = \frac{1}{2\pi \times R_4 \times 4000Hz} = \frac{1}{2\pi \times R_9 \times 4000Hz} \quad (4\text{-}6)$$

où 4000Hz est la bande passante approximative souhaitée.

TABLEAU 4-8 – VALEUR DES COMPOSANTS POUR L'AMPLIFICATEUR D'ENTREE AUDIO

Entrée du module	MIC_HF
GA	+30.9 dB
gain	35.2
R4 = R9	180 kΩ
R3 = R7	5.1 kΩ
C7=C10	220 pF

Ainsi, avec les valeurs des composants mentionnées ci-dessus, on obtient un amplificateur adapté à nos besoins.

Sortie audio

Ce montage est destiné à un haut-parleur de type « oreillette ». Il a à son entrée la sortie EAR_MT du module GM862-GPS.

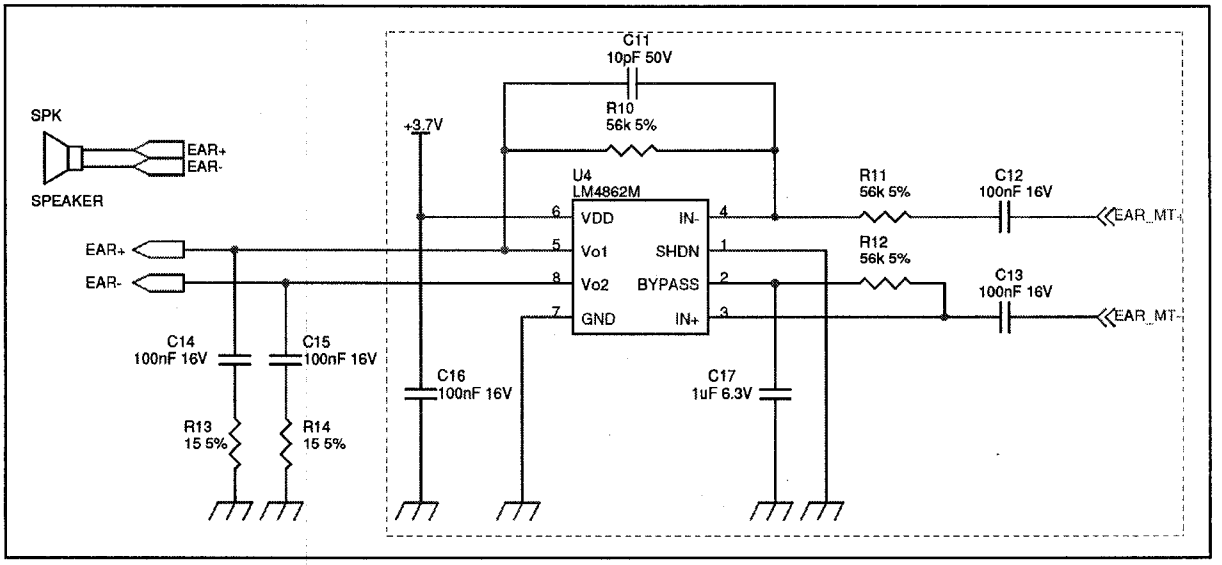


Figure 4-16 – Schéma électrique de la sortie audio

Le haut-parleur doit avoir une impédance nominale de 16Ω. Il convient d'éloigner autant que possible le microphone et le haut-parleur, afin d'éviter autant que possible l'Effet Larsen[1].

Tests

Un simple test consiste à connecter les interfaces audio au module cellulaire, à piloter ce dernier depuis un émulateur de terminal sur l'ordinateur et à effectuer un appel. On peut également tester les circuits en connectant la sortie de l'amplificateur du microphone à l'entrée de l'amplificateur du haut-parleur.

[1] Wikipedia – Effet Larsen – http://fr.wikipedia.org/wiki/Effet_Larsen

4.2.6 Alimentation

Problématique

La batterie utilisée est celle décrite dans la section 3.2.6, une batterie Lithium Ion 3.7V 2200mAh de chez Varta.

Cependant, les tensions d'alimentation des autres composants de la plateforme sont problématiques. En effet, le microcontrôleur LPC2106, l'afficheur VFD2041 ainsi que les composants qui serviront à interfacer le clavier ont besoin d'être alimentés en 5V, alors que la batterie fournit une tension proche de 3.7V (légèrement variable selon son état de charge), tension imposée par le chargeur du module cellulaire.

Obtention du 5V

Il sera donc nécessaire d'utiliser le convertisseur DC/DC 5V VPack proposé par SparkFun Electronics[1], qui permet, à partir d'une tension d'entrée comprise entre 1.1 et 4.5V, d'obtenir 5V en sortie. Ce convertisseur pouvant fournir un courant de 300mA, 2 de ces composants seront requis afin d'alimenter d'une part l'afficheur, et d'autre part le microcontrôleur et les composants servant à l'interfaçage du clavier.

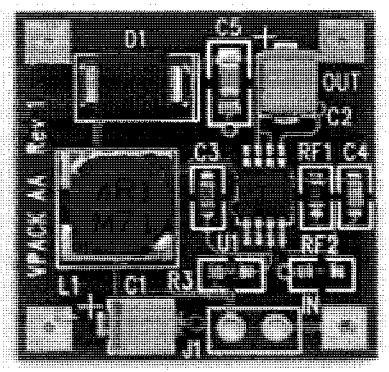


Image SparkFun Electronics

Figure 4-17 – Photo du convertisseur DC/DC 5V Vpack

[1] Sparkfun Electronics – Module DC/DC 5V DPack – http://www.sparkfun.com/commerce/product_info.php?products_id=8290

Il est à noter que si le VPack doit fournir 5V et 250mA en sortie à partir de 3.7V, on mesurera à l'entrée du VPack un courant de 420mA environ, selon la formule suivante, extraite de l'équation (2) de la fiche technique du produit fournie par Sparkfun [SPARKFUN, 2008] :

$$I_{IN} = \frac{1}{0.8} \times \frac{V_{OUT}}{V_{IN}} \times I_{OUT} \qquad (4\text{-}7)$$

Consommations électriques

Afin de concevoir l'alimentation, il est nécessaire de connaitre les consommations électriques de chaque section du montage. Aussi, voici les valeurs calculées et les valeurs mesurées des courants du système :

TABLEAU 4-9 – TENSIONS ET COURANTS D'ALIMENTATION DES PRINCIPAUX COMPOSANTS

Section	Tension d'alim	Courant mesuré
Microcontrôleur LCP2106	5V	60mA
Clavier et interfaces	5V	100mA
Afficheur VFD2041	5V	250mA
Module cellulaire GM862-GPS et interfaces	3.7V	35/170mA (*iddle*/en communication)
Total	-	445/640mA

On souhaite donc alimenter en 5V le microcontrôleur LPC2106, le clavier et ses interfaces, ainsi que l'afficheur VFD2041. Un convertisseur DC/DC tel que décrit dans la section précédente peut fournir 5V/250mA. Il sera donc nécessaire d'employer 2 de ces modules.

Ainsi :

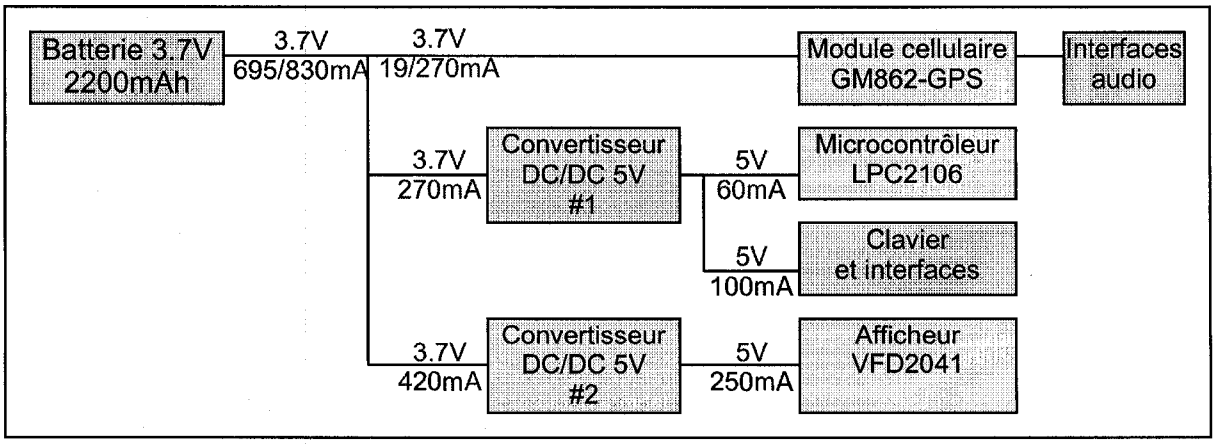


Figure 4-18 –Répartition de la consommation en courant du circuit

Protection

Afin de protéger des surtensions et des inversions de tension, on utilise une diode TVS et un fusible auto-réarmable (*polyswitch*).

Finalement, le schéma électrique est le suivant :

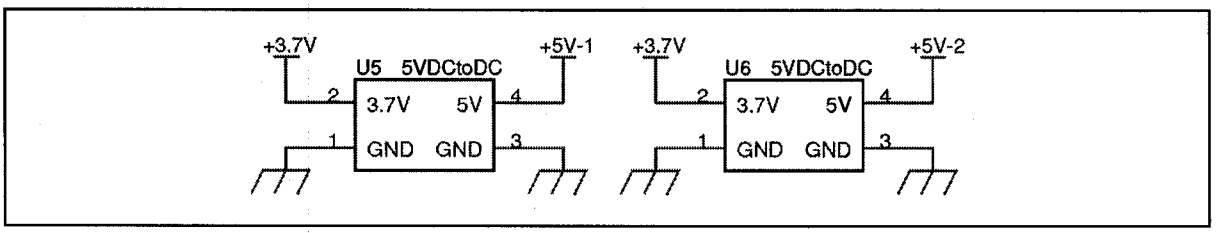


Figure 4-19 – Schéma électrique de l'alimentation

4.3 Circuit imprimé

Avoir le montage sur *breadboard* et *veroboard* n'est pas très pratique et peu esthétique, comme le montre la photo ci-contre.

Figure 4-20 – Photo du montage avant la fabrication du PCB

Aussi, à partir du schéma électrique (voir le schéma complet en annexe), on réalise le PCB (*Printed Circuit Board*) de l'ensemble. Le clavier et l'écran déterminent les dimensions finales du PCB : 10x16cm.

Voici la forme générale de l'ensemble :

- L'afficheur et le clavier forment la couche supérieure. Ils sont vissés, par l'intermédiaire d'entretoises, au PCB ;
- Le microcontrôleur ainsi que les interfaces du clavier sont directement fixés SUR le PCB, par l'intermédiaire de supports ;

- Enfin, le module cellulaire ainsi que les 2 convertisseurs DC/DC et la batterie sont directement fixés SOUS le PCB, par l'intermédiaire de supports ;
- Bien entendu, les composants externes tels que résistances et capacités sont proches des composants auxquels ils sont rattachés, mais ne figurent pas sur le dessin ci-dessus.

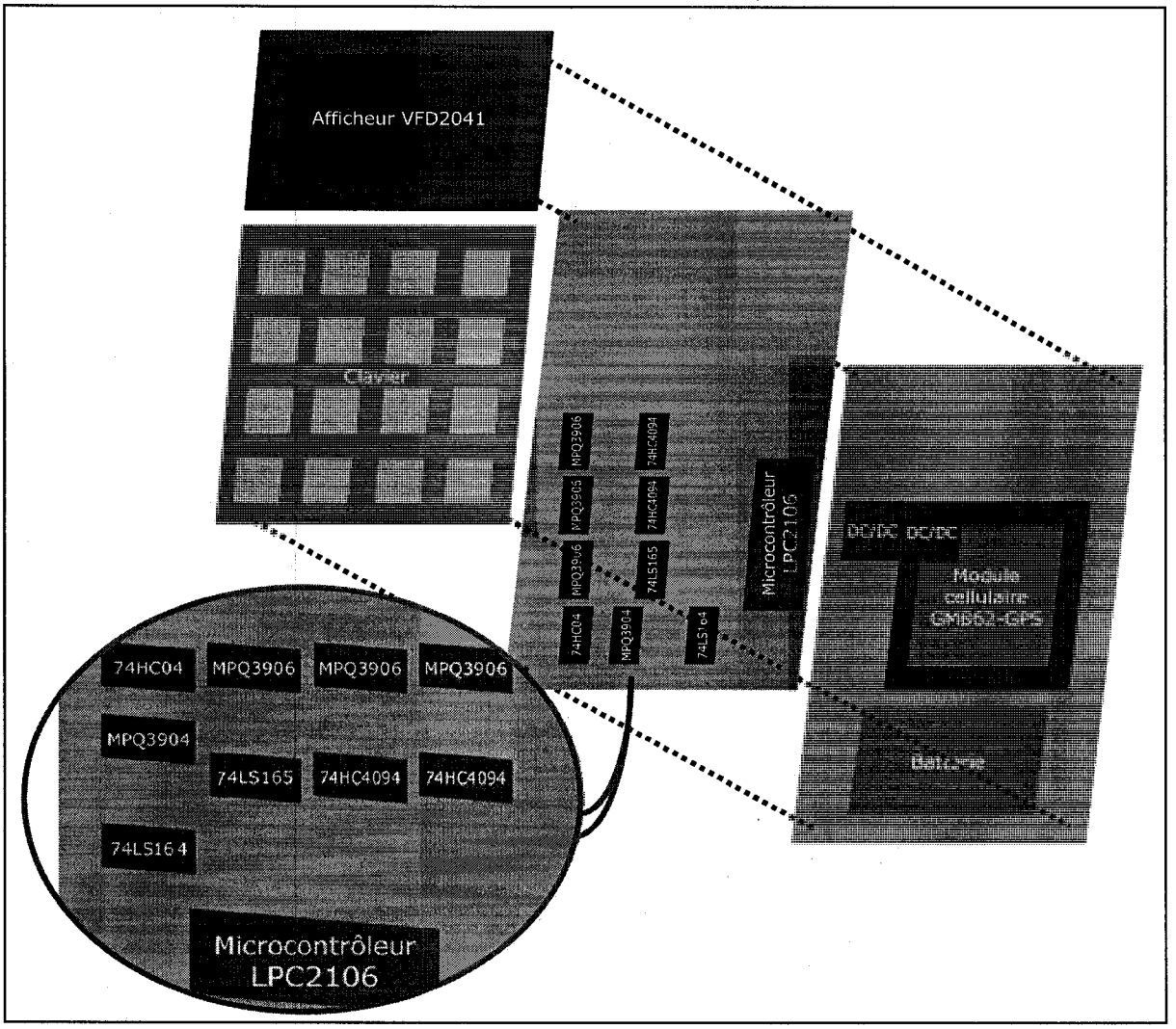


Figure 4-21 – Schéma de l'empilement des composants sur le PCB

Le PCB est réalisé sous Cadence OrCAD Layout Plus 10.2. Une empreinte est réalisée pour les composants n'en ayant pas directement, comme le clavier, le module cellulaire, le microcontrôleur, etc...

Voici l'aspect final du PCB avant réalisation :

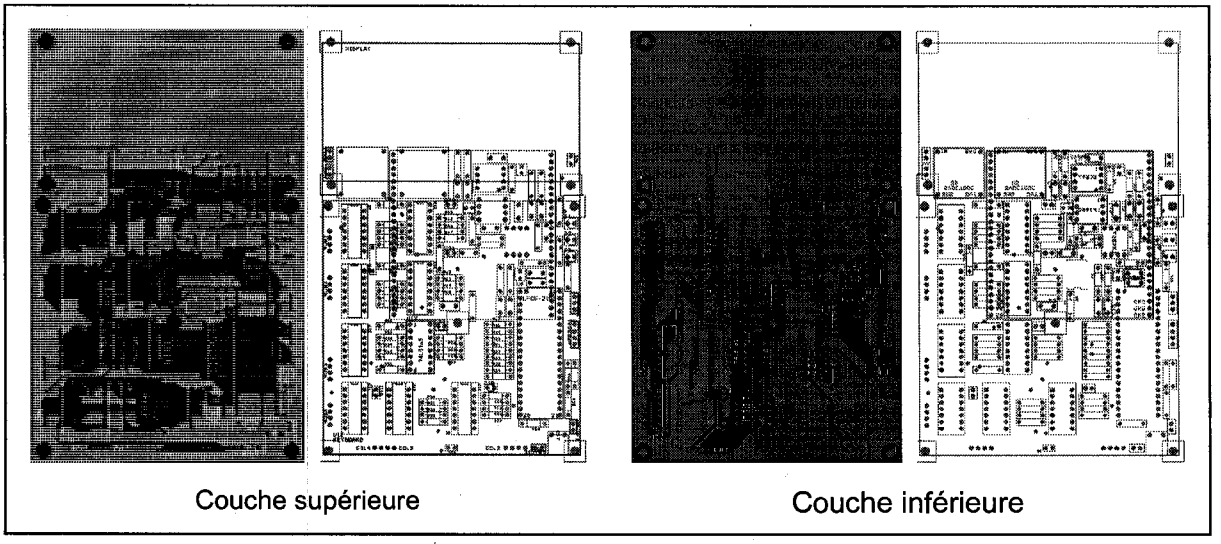


Figure 4-22 – Aspect final du PCB sous OrCAD Layout

Au final, le PCB réalisé par le service technique du département de génie électrique et de génie informatique a l'aspect suivant :

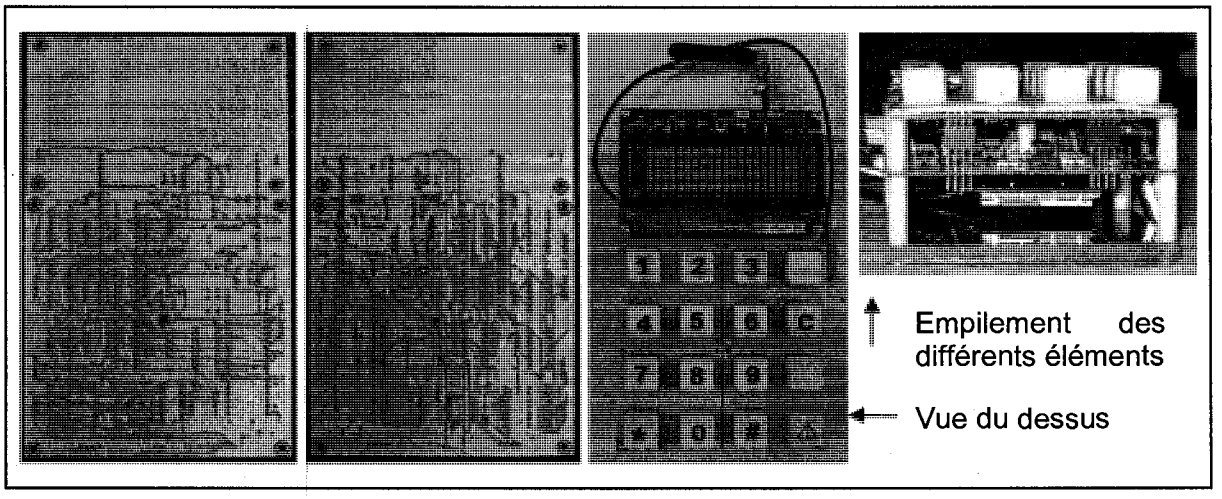


Figure 4-23 – Photos du PCB avant et après assemblage des composants

4.4 Versions futures

La plateforme, dans son état actuel, peut être utilisable expérimentalement mais devra être améliorée pour être éventuellement industrialisable et commercialisable, notamment en termes de poids, de consommation électrique, et d'ergonomie. Ainsi, il conviendra d'apporter au moins ces modifications :

- Choisir une batterie d'un format standard et facilement trouvable dans le commerce ;
- Intégrer les circuits externes propres à l'afficheur, au microcontrôleur et au cellulaire directement au PCB ;
- Choisir des composants montés en surface, notamment pour les interfaces du clavier ;
- Utiliser des circuits basse consommation (comme la famille 74HC) au lieu de TTL (comme la famille 74LS) ;
- Prévoir un boîtier esthétique, résistant et manipulable facilement.

CHAPITRE 5.

MISE EN ŒUVRE LOGICIELLE

5.1 Architecture

Les étapes de développement logiciel ont été les suivantes :

- A chaque ajout d'un nouveau composant matériel, le développement des pilotes (bas niveau) et d'une légère application de test a été effectué ;
- Une fois tous les composants intégrés, une rapide application a été réalisée en ajoutant tous les programmes de test ;
- Puis, tout a été réorganisé afin d'intégrer l'OS temps réel µC/OS-II. Les pilotes bas-niveau ainsi que les fonctions d'interface avec les pilotes bas-niveau ont été réutilisées, mais l'architecture du système est différente.

C'est cette architecture finale qui sera présentée ici.

5.2 Couches de l'architecture logicielle

La Figure 5-1 présente l'organisation en couches de l'architecture logicielle. On peut y observer l'organisation en couches du système. En effet, en partant du bas, se trouve la couche bas-niveau avec les pilotes d'interface pour le matériel. Un peu plus haut, se trouve la couche intermédiaire, ou *middleware*, constituant 3 boîtes à outils. Ainsi, l'application, située sur la couche supérieure, peut faire appel au matériel par le biais de fonctions pratiques à utiliser.

L'application fait ainsi appel aux boites à outils, qui elles-mêmes font appel aux pilotes bas niveau, pour utiliser le matériel (l'afficheur, le clavier et le module cellulaire ici).

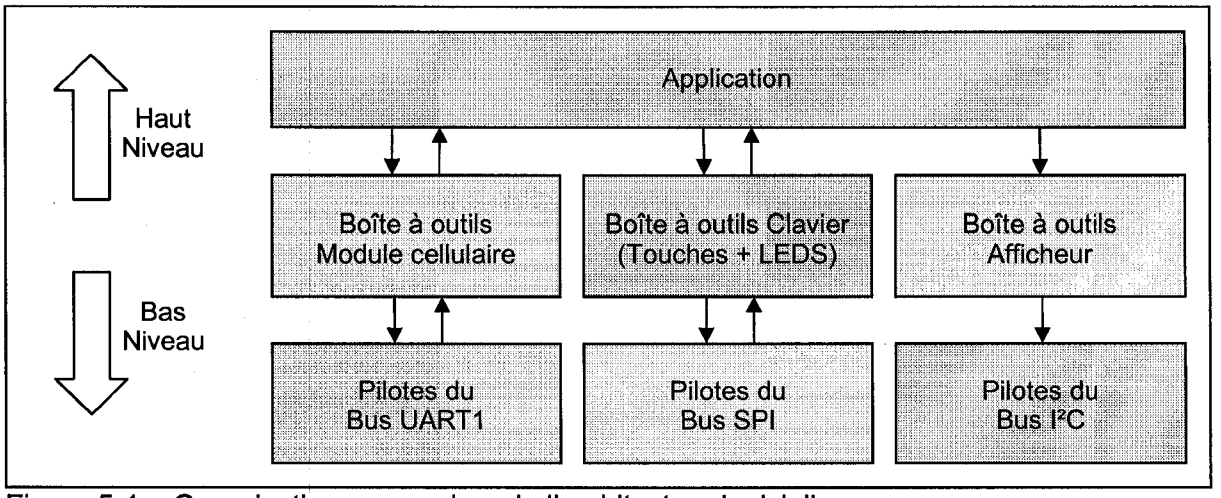


Figure 5-1 – Organisation en couches de l'architecture logicielle

5.3 Développement des pilotes bas-niveau

Les prototypes des principales fonctions utilisées sont disponibles en Annexe, section A.1.1.

5.3.1 Pilotes de l'UART0/1 (port série)

Les pilotes utilises pour l'UART proviennent de RO Software et ont été adaptés et diffusés par Martin Thomas[1]. Ils permettent d'utiliser l'UART par interruption sur la réception, sur la transmission, sur les 2 ou sur aucune. Les détails sur l'utilisation des UART sur le microcontrôleur LPC2106 sont consultables dans le Manuel d'Utilisateur [NXP, 2003um], pages 85 et suivantes.

Initialisation

Pour l'initialisation des UARTs 0 et 1, les fonctions `uart0Init()` et `uart1Init()` sont nécessaires afin de fixer les paramètres suivants :

[1] ARM-Projects – http://www.siwawi.arubi.uni-kl.de/avr_projects/arm_projects/index.html#lpc_uart_irq

TABLEAU 5-1 – PARAMETRES D'INITIALISATION DES UARTS

`UART_BAUD(115200)`	115 200 bauds
`UART_8N1`	8 bits, pas de parité, 1 bit de STOP
`UART_FIFO_8`	FIFO déclenchée à 8 caractères

Utilisation

Sont à la disposition de l'utilisateur les fonctions principales suivantes, dont les prototypes figurent en Annexe : `uart0Putch()`, `*uart0Puts()`, `uart0Getch()`, `uart0ISR()`.

Tests

Les pilotes de l'UART peuvent être testés avec un émulateur de terminal sur un ordinateur. Pour cela, on utilise un composant CP2102[1] qui, à partir des signaux Rx et Tx de l'UART0 du microcontrôleur, permet de relier la plateforme à l'ordinateur par le port USB. C'est d'ailleurs de cette manière que le code est chargé dans le microcontrôleur.

Ainsi, on a la possibilité de tester les écritures `uart0Putch()`, `*uart0Puts()` et les lectures `uart0Getch()`, de façon indépendante du système.

5.3.2 Pilotes du bus SPI

Les pilotes du bus SPI ont été développés par l'auteure de ce mémoire. Les détails sur l'utilisation du bus SPI sur le microcontrôleur LPC2106 sont consultables dans le Manuel d'Utilisateur [NXP, 2003um], pages 123 et suivantes.

Initialisation

Pour l'initialisation du bus SPI, la fonction `InitSPI()` est nécessaire afin de fixer les paramètres suivants:

TABLEAU 5-2 – PARAMETRES D'INITIALISATION DU BUS SPI

`PINSEL0`	Définition des broches d'entrée / sortie (GPIO) comme fonction SPI
`SPI_SPCCR`	SPI Clock Counter Register : (voir détails plus bas)
`SPI_SPCR`	SPI Control Register : (voir détails plus bas)

[1] Sparkfun Electronics – CP2102 – http://www.sparkfun.com/commerce/product_info.php?products_id=198

Première étape, on attribue aux broches du microcontrôleur les fonctions de SPI selon [NXP, 2003um], page 78 table 48 :

TABLEAU 5-3 – CONFIGURATION DES BROCHES D'ENTREE-SORTIE POUR LE BUS SPI

PINSEL0	Nom de la broche	Fonction quand PINSEL0 = 01
09 : 08	P0.4	SCK : *Serial ClocK* (SPI)
11 : 10	P0.5	MISO : *Master In Slave Out* (SPI)
13 : 12	P0.6	MOSI : *Master Out Slave In* (SPI)
15 : 14	P0.7	SSEL : *Slave SELect* (SPI)

Puis on configure le registre *SPI Clock Counter Register*. On souhaite que la vitesse de communication du bus SPI soit maximale. De ce fait, le registre SPI_SPCCR est initialisé à 0xFE, valeur maximale car devant être paire, selon [NXP, 2003um], page 130.

Ainsi, la fréquence du bus SPI peut être calculée de la façon suivante :

$$SPIrate(Hz) = \frac{PCLK}{SPCCR} = \frac{\frac{CCLK}{VPB}}{SPCCR} = \frac{\frac{FOSC \times M}{VPB}}{SPCCR} = \frac{\frac{14.7456 \cdot 10^6 \times 4}{1}}{254} = 232.2 \cdot 10^3 = 232.2kHz \quad (5\text{-}1)$$

ou $SPIrate(s) = 4.31\mu s$. On suppose ici que l'on n'utilise pas la PLL (*Phase-Locked Loop*) et que l'horloge du microcontrôleur (PCLK) est celle du cristal soit 14,7 MHz.

Enfin, on configure le registre *SPI Control Register*, de la manière suivante :

TABLEAU 5-4 – CONFIGURATION DU REGISTRE SPI CONTROL REGISTER

Bit de SPCR	Fonction	Valeur choisie
2 : 0	Réservé	0
3	CPHA : *Clock Phase Control* Si 0 : données échantillonnées sur 1er front de SCK Si 1 : données échantillonnées sur 2ème front de SCK	0
4	CPOL : *Clock Polarity Control* Si 0 : SCK active à l'état haut Si 1 : SCK active à l'état bas	0
5	MSTR : *Master mode select* Si 0 : SPI en mode esclave Si 1 : SPI en mode maître	1
6	LSBF : *Least Significant Bit First* Si 0 : MSB est transmis en premier Si 1 : LSB est transmis en premier	0
7	SPIE : *Serial Peripheral Interrupt Enable* Si 0 : pas d'interruption Si 1 : une interruption matérielle est générée à chaque fois que les bits SPIF ou MODF sont activés	0

Utilisation

La fonction de lecture-écriture sur le port SPI a été divisée en 2 fonctions distinctes, `ReadWriteSPI()` et `ReadWriteSPI_fin()`, pour des raisons qui seront détaillées dans la section concernant les LEDs du clavier.

Tests

Il n'est pas possible d'interfacer facilement le port SPI du microcontrôleur avec un ordinateur, à l'image de l'UART. Aussi, pour tester le bon fonctionnement du port SPI, il faut connecter un 74LS164, envoyer une trame MOSI et un signal CS0 adapté, et observer en sortie les trames à l'oscilloscope. Ceci teste l'écriture sur le port SPI.

A l'inverse, pour la lecture, il suffit d'ajouter un 74LS165 au montage précédent, de relier les sorties du 164 aux entrées du 165, et de vérifier que ce qu'on lit en MISO correspond à ce qu'on a écrit en MOSI.

5.3.3 Pilotes du bus I²C

Les pilotes du bus I²C sont inspirés de Pierre Seguin, modifiés par Owen Mooney[1] et par l'auteure de ce mémoire. Les détails sur l'utilisation du bus I²C sur le microcontrôleur LPC2106 sont consultables dans le Manuel d'Utilisateur [NXP, 2003um], pages 111 et suivantes.

Initialisation

Pour l'initialisation du port I²C, la fonction `InitI2C()` est nécessaire afin de fixer les paramètres suivants :

TABLEAU 5-5 – PARAMETRES D'INITIALISATION DU PORT I²C

`PINSEL0`	Définition des broches d'entrée / sortie (GPIO) comme fonction I²C
`I2C_I2SCLL`	*I²C SCL Duty Cycle Register Low Half Word* : (voir détails plus bas)
`I2C_I2SCLH`	*I²C SCL Duty Cycle Register High Half Word* : (voir détails plus bas)
`I2C_I2CONSET`	*I²C Control Set Register* : (voir détails plus bas)

[1] LPC2000 Yahoo Group– http://tech.groups.yahoo.com/group/lpc2000 - "Files" - "I2c_w_error.c"

L'attribution des broches du microcontrôleur se fait de la manière suivante, selon [NXP, 2003um], page 78 table 48 :

Tableau 5-6 – Configuration des broches d'entrée-sortie pour le bus I²C

PINSEL0	Nom de la broche	Fonction quand PINSEL0 = 01
5 : 4	P0.2	SCL : *Serial CLock* (I²C)
7 : 6	P0.3	SDA : *Serial Data* (I²C)

Puis on configure les registres *I²C SCL Duty Cycle Register High Half Word* et *I²C SCL Duty Cycle Register Low Half Word*. Le bus I²C servant uniquement à communiquer avec l'afficheur, il convient de décider à quelle vitesse on souhaite cette communication. L'afficheur supporte une vitesse allant jusqu'à 400 Kbps, selon [MO, 2006] page 16, mais dans notre cas, on a choisi la vitesse par défaut de 19.2kHz, selon [MO, 2006] table 2 page 15. De ce fait, les registres à configurer sont tels que :

$$I2SCLL + I2CLH = \frac{PCLK}{I^2Crate_{(Hz)}} = \frac{\frac{CCLK}{VPB}}{I^2Crate_{(Hz)}} = \frac{\frac{FOSC \times M}{VPB}}{I^2Crate_{(Hz)}} = \frac{\frac{14.7456 \cdot 10^6 \times 4}{1}}{19200} = 2 \times 1536 \quad (5\text{-}2)$$

Ainsi, I2C_I2SCLL = I2C_I2SCLH = 1536. (5-3)

Finalement, pour le registre *I²C Control Set Register*, on le configure de la façon suivante :

TABLEAU 5-7 – CONFIGURATION DU REGISTRE I²C CONTROL SET REGISTER

Bit de I2CONSET	Fonction	Valeur choisie
1 : 0	Réservé	0
2	*Assert Acknowledge Flag* Si 0 : pas d'utilisation de ce drapeau Si 1 : utilisation	0
3	*I²C Interrupt Flag* Si 0 : pas d'utilisation de ce drapeau Si 1 : utilisation	0
4	*STOP Flag* Si 0 : pas d'utilisation de ce drapeau Si 1 : utilisation	0
5	*START Flag* Si 0 : pas d'utilisation de ce drapeau Si 1 : utilisation	0
6	*I²C Interface Enable* Si 0 : fonctions I²C désactivées Si 1 : fonctions I²C activées	1
7	Réservé	0

Utilisation

Sont à la disposition de l'utilisateur les fonctions suivantes : `SendAddressI2C()`, `ReadI2C()`, `WriteI2C()`, `WriteStringI2C()` et `StopI2C()`.

Tests

Un test simple consistera à écrire un octet sur le port I²C, et vérifier à l'aide d'un oscilloscope que la trame sur la broche SDA (avec les résistances de *pull-up*) correspond effectivement à l'octet envoyé.

5.4 Développement des boîtes à outils intermédiaires

5.4.1 Boîte à outils relative au module cellulaire

Afin de communiquer avec le module cellulaire, deux broches d'entrée/sortie du microcontrôleur seront nécessaires, en plus de l'UART1.

Initialisations

L'initialisation du module cellulaire nécessite les étapes suivantes :

TABLEAU 5-8 – ÉTAPES D'INITIALISATION DE L'AFFICHEUR

Étape	Fonction appelée
Initialisation de l'UART1	`uart1Init ()`
Initialisation des broches d'entrée/sortie vers le cellulaire	`PINSEL1, IODIR, IOSET`

Ces étapes sont regroupées dans la fonction :

```
void InitCell(void)
```

Voici le détail des initialisations des différentes broches :

TABLEAU 5-9 – INITIALISATION DES CONNEXIONS VERS LE MODULE CELLULAIRE

Connexion	Fonction	Broche microcontrôleur	Broche cellulaire	Initialisation
Tx1	UART1	P0.8	TX (#16)	`uart1Init()`
Rx1		P0.9	RX (#44)	
ON/OFF	Allume / éteint le cellulaire	(bouton poussoir)	ON/OFF (#34)	(sans)
PWRMON	Surveille le statut allumé / éteint du cellulaire	P0.27	PWRMON (#36)	Entrée `IODIR&=0xF7FFFFFF`

Gestion de l'alimentation

Le module cellulaire s'allume à l'aide d'un appui d'au moins une seconde sur le bouton-poussoir dédié. Cela correspond à ce qui est mentionné dans [TELIT, 2007hw], aux pages 19 à 23 : la broche ON/OFF du module cellulaire doit être maintenue à l'état bas pendant au moins 1 seconde avant de retourner à l'état haut. Cependant, il est souhaitable de pouvoir éteindre et connaitre le statut allumé/éteint du module cellulaire. Pour cela, les fonctions suivantes seront utiles :

```
void TurnOffCell(void)
INT8U GetPwrMonCell(void)
```

Avant d'éteindre le module, on utilise donc la fonction `GetPwrMonCell()` qui permet de vérifier si le module est déjà allumé ou déjà éteint en lisant l'état de la broche `PWRMON` du module cellulaire.

La fonction `TurnOffCell()`, elle, réalise un « Software Shutdown » puisqu'elle demande logiciellement au module de s'éteindre, grâce à une commande AT spécifique.

Envoi de commandes

Le module cellulaire contient un interpréteur Python. Cependant, il a été choisi de le commander par communication série (port UART). Ainsi, il est possible de lui envoyer des commandes de type « AT » [TELIT, 2007at] et d'obtenir une réponse à cette commande.
Parmi les commandes principales, le dispositif doit pouvoir :

- Réaliser toutes les initialisations nécessaires au bon fonctionnement du module ;
- Appeler un numéro de téléphone ;
- Décrocher lors d'un appel entrant ;
- Raccrocher lors d'un appel en cours ;
- Connaitre le statut sur le réseau GSM ;

- Allumer, éteindre et connaitre la position du GPS ;
- Envoyer un SMS préprogrammé.

Pour cela, les fonctions suivantes seront nécessaires :

```
void InitConfigCell(void)
void DialGSM (unsigned char *pPhoneNb)
void HangUpGSM(void)
void AnswerGSM(void)
INT8U CheckIncomingCallGSM(void)
void CheckNetworkStatusGSM(void)
void TurnOnGPS(void)
void TurnOffGPS(void)
void GetPosGPS(void)
void SendSMS(INT8U SmsID)
```

L'ensemble du traitement des commandes AT ne sera pas détaillé ici puisque toutes les informations nécessaires sont disponibles à la référence [TELIT, 2007at].

Envoi d'une commande et réception de la réponse

La fonction `askGM862()` est chargée de dialoguer avec le module cellulaire. En effet, c'est elle qui envoie une commande et en attend la réponse. Cette opération se déroule en plusieurs étapes :

- Prise d'un sémaphore afin de pouvoir accéder au module cellulaire (la gestion des sémaphores sera détaillée ultérieurement, dans la section 5.5.3) ;
- Envoi de la chaine de caractères correspondant à la commande, suivie du caractère « Retour Chariot », ou "*Carriage Return*", ou « `\r` » ;
- Attente, caractère par caractère, des informations renvoyées par le module cellulaire sur le port série, jusqu'à détection du caractère signalant la fin de transmission « Retour Chariot » ;
- Libération du sémaphore d'accès au cellulaire.

La réponse est stockée dans une chaine de caractères pointée par `*response` et réinitialisée à 0 avant chaque attente de réponse. La portion de code pertinente de la fonction `askGM862()` figure ci-dessous :

```
OSSemPend(AccessToCellSem, 0, &err);  // Ask for semaphore (no timeout)
uart1Puts(request);                   // Send command to GM862 with UART
uart1Puts("\r");
for (i=0; i<40; i++)
      *(response+i) = 0;              // Reinit response
i = 0;
do{
      ch = uart1Getch();              // Get character from UART1
      if (ch != -1){                  // If a character is available on UART1
            if (ch == 10) ch = LF_CHAR;
            if (ch == 13) ch = CR_CHAR;
            *(response+i) = ch;       // Store character in response
            i++;
      }
}while(*(response+i-1) != CR_CHAR || i == 40);
OSSemPost(AccessToCellSem);           // Release semaphore
```

Ainsi, il est possible de dialoguer avec le module cellulaire. L'interprétation des réponses se fait dans les fonctions ayant appelé la fonction `askGM862()`.

La fonction `CheckIncomingCallGSM()` fonctionne de façon similaire, car elle permet de vérifier périodiquement si le cellulaire est en train de recevoir un appel. Dans ce cas, on cherche à détecter le message non sollicité « `RING\r` », envoyé par le cellulaire alors qu'aucune commande ne lui a été envoyée.

Initialisation de la configuration du cellulaire

Voici la configuration utilisée lors de l'initialisation, par la fonction `InitConfigCell()` :

TABLEAU 5-10 – INITIALISATION DES PARAMETRES INTERNES DU MODULE CELLULAIRE

Rôle	Commande
Désactiver l'écho (la répétition des commandes envoyées au cellulaire)	ATE0
Définir le format de réponse numérique (et non du texte)	ATV0
Définir les codes de résultat « silencieux »	ATQ0
Désactiver les codes de résultats non sollicités lors de la connexion au réseau	AT+CREG=1
Vérifier l'état du réseau GSM	AT+CREG?
Vérifier la force du signal GSM	AT+CSQ
Définir l'appareil en mode voix (par opposition au mode data)	AT+FCLASS=8
Définir les canaux d'entrée / sortie de l'audio (HF=1, MT=2)	AT+CAP=1

Ces configurations sont envoyées successivement au cellulaire via la fonction `askGM862()` évoquée précédemment.

Tests

Les fonctions réalisées dans cette boîte à outils sont testables en utilisant le port série pour afficher les réponses du module cellulaire. Le cellulaire étant relié par l'UART1 du microcontrôleur, on peut recopier ses réponses sur l'UART0 et les afficher sur l'émulateur de terminal.

5.4.2 Boîte à outils relative au clavier

Initialisations

L'initialisation du clavier nécessite les étapes suivantes :

TABLEAU 5-11 – ÉTAPES D'INITIALISATION DE L'AFFICHEUR

Étape	Fonction appelée
Initialisation du bus SPI	`InitSPI()`
Initialisation du Timer1	`Tmr_Timer1Init()`
Initialisation de la structure contenant les informations sur les 16 touches	`InitKeyStruct()`

Ces étapes sont regroupées dans la fonction :

```
void InitKeyboard(void)
```

Initialisation du Timer1

En ce qui concerne l'utilisation du Timer1 : lors d'une écriture-lecture sur le bus SPI, le fait de charger une donnée dans le registre `SPI_SPDR` suffit à démarrer l'horloge `SCK` et l'envoi des données. Cependant, pour la détection de l'appui d'une touche du clavier, on a besoin d'activer ou de désactiver les signaux de contrôle `CS0` et `CS3` au bon moment. Pour cela, on utilise le Timer1 afin de connaitre à quel moment on se situe dans la procédure.

Une trame SPI dure 4.3µs x 8 = 34.4µs. Si on consulte la *datasheet* du composant 74LS165, un signal `CLOCK INHIBIT`, qui sera à 0 tout le temps des échanges (il s'agit du signal `/CS0` dans notre cas), sera nécessaire. De même qu'un signal `SHIFT/LOAD`, qui, lorsqu'il passera à 0, permettra au composant 74LS165 de recopier en sortie sous forme de trame série (`OUTPUT` Q_H) les signaux parallèles qu'il a en entrée (`P0..P7`). Ce signal est assuré par `CS3`, qui, à la fin de chaque écriture-lecture SPI, se mettra à 0.
Voici un extrait de la *datasheet* en question :

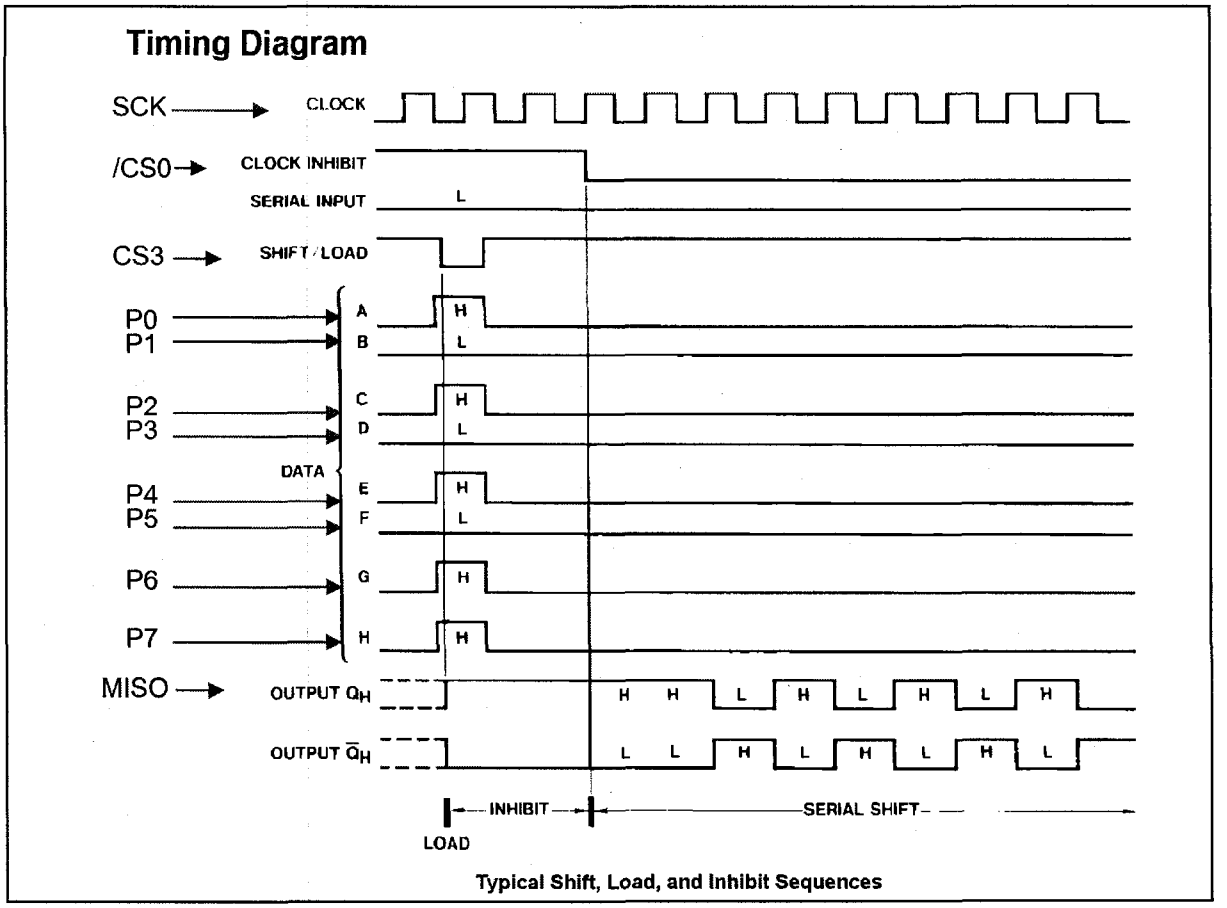


Figure 5-2 – Chronogramme d'utilisation du 74LS165

Et, pour illustrer, voici les signaux correspondant dans notre montage :

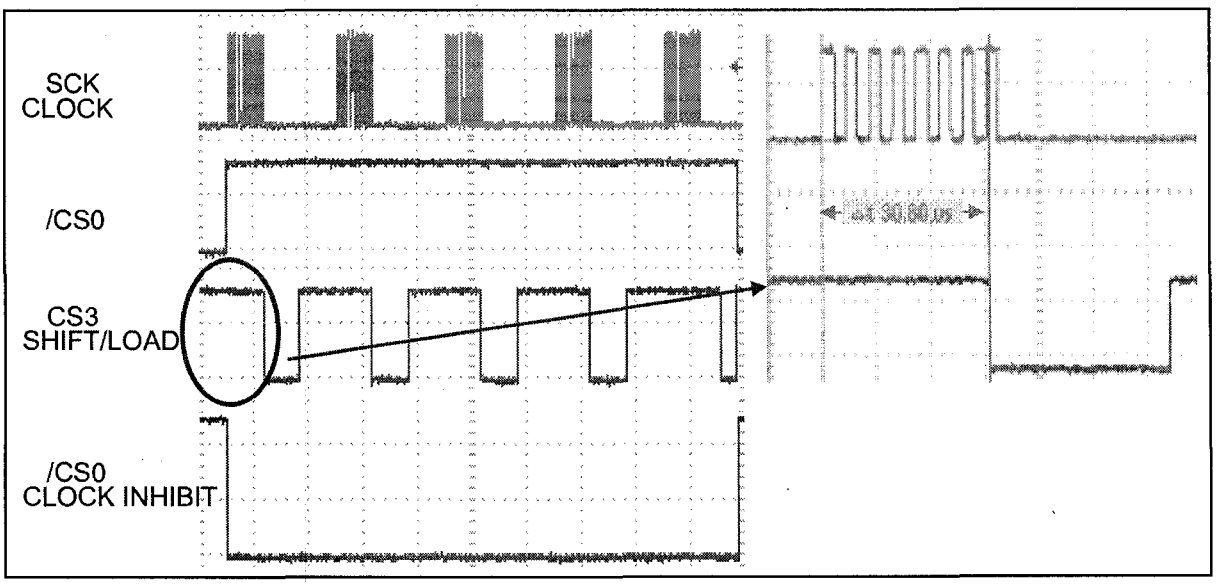


Figure 5-3 – Chronogramme d'utilisation des signaux de sélection pour le clavier

Ainsi, on utilisera un Timer1 d'une période de 33µs permettant le passage à 0 de CS3 à la fin de la trame SCK. Le Timer1 est donc initialisé de la manière suivante :

```
VICIntSelect&= ~(1 << VIC_TIMER1);          // Enable interrupts
VICVectAddr3 = (INT32U)Tmr_Timer1ISR_Handler;  // Set the vector address
VICVectCntl3 = 0x20 | VIC_TIMER1;           // Enable vectored interrupts
VICIntEnable = (1 << VIC_TIMER1);           // Enable Interrupts

TIMER1_TCR  = 0;                            // Disable timer 1.
TIMER1_PC   = 0;                            // Prescaler is set to no division

TIMER1_MR0  = BSP_CPU_ClkFreqPeripheral()/TIMER1_PERIOD;
TIMER1_MCR  = 3;                            // Interrupt on MR0 (reset TC)

TIMER1_CCR  = 0;                            // Capture is disabled
TIMER1_EMR  = 0;                            // No external match output
TIMER1_TCR  = 1;                            // Enable(0) or Disable(1) timer 1
```

avec TIMER1_PERIOD = 7500, correspondant à une période mesurée de 33µs.

Initialisation de la structure de touches

En ce qui concerne l'initialisation de la structure contenant les informations sur les 16 touches, on commence par définir la structure de données KEY_STR contenant les informations relatives à une touche :

```
typedef struct {
      INT8U ID;
      INT8U Column;            // 1, 2, 3, 4
      INT8U Row;               // 1, 2, 3, 4
      INT8U Color;             // 1 (red), 2 (blue), 3 (green)
      INT8U LedOn;             // 1 (yes), 2 (no)
      unsigned char Symbol;    // '1', '2', ...
} KEY_STR;
```

Puis, on crée un tableau contenant les informations relatives à toutes les touches :

TABLEAU 5-12 – INITIALISATION DE LA STRUCTURE DES TOUCHES DU CLAVIER

Key[]	ID	Column	Row	Color	LedOn	Symbol
Key[0]	1	1	1	2 (bleu)	1 (oui)	'1'
Key[1]	2	2	1	2 (bleu)	1 (oui)	'2'
Key[2]	3	3	1	2 (bleu)	1 (oui)	'3'
Key[3]	4	4	1	3 (vert)	1 (oui)	'K'
Key[4]	5	1	2	2 (bleu)	1 (oui)	'4'
Key[5]	6	2	2	2 (bleu)	1 (oui)	'5'
Key[6]	7	3	2	2 (bleu)	1 (oui)	'6'
Key[7]	8	4	2	1 (rouge)	1 (oui)	'C'
Key[8]	9	1	3	2 (bleu)	1 (oui)	'7'
Key[9]	10	2	3	2 (bleu)	1 (oui)	'8'
Key[10]	11	3	3	2 (bleu)	1 (oui)	'9'

Key[11]	12	4	3	1 (rouge)	0 (non)	'U'
Key[12]	13	1	4	1 (rouge)	0 (non)	'L'
Key[13]	14	2	4	2 (bleu)	1 (oui)	'0'
Key[14]	15	3	4	1 (rouge)	0 (non)	'R'
Key[15]	16	4	4	1 (rouge)	0 (non)	'D'

Ceci est effectué par les fonctions :

```
void InitKeyStruct (void)
void AssignKeyStruct (
        INT8U index,
        INT8U kID,
        INT8U kColumn,
        INT8U kRow,
        INT8U kColor,
        INT8U kLedOn,
        unsigned char kSymbol)
```

Détection de l'appui d'une touche

La détection de l'appui d'une touche se fait en plusieurs étapes :

- Dans la fonction `ReadKeyboard()`, le démarrage du Timer1 entraine un appel à l'ISR (*Interrupt Service Routine*) `Tmr_Timer1ISR_Handler()`, qui elle-même appelle la fonction `KeyboardProcessSPI();`
- S'ensuivent les 5 écritures-lectures sur le port SPI, qui correspondent aux étapes décrites à la section 4.2.3.

Ainsi, on rappelle les 5 étapes :

TABLEAU 5-13 – ORGANISATION DES TRAMES SPI POUR LA DETECTION DE L'APPUI D'UNE TOUCHE

Cycle (SCK)	Trame 1	Trame 2	Trame 3	Trame 4	Trame 5
Écriture (MOSI)	Ligne 1	Ligne 2	Ligne 3	Ligne 4	~~(à ignorer)~~
Lecture (MISO)	~~(à ignorer)~~	Ligne 1	Ligne 2	Ligne 3	Ligne 4

Les données stockées à chaque étape sont les suivantes :

TABLEAU 5-14 – ORGANISATION DES DONNEES POUR LA DETECTION DE L'APPUI D'UNE TOUCHE

Cycle (SCK)	Trame 1	Trame 2	Trame 3	Trame 4	Trame 5
Donnée stockée	~~(à ignorer)~~	ByteRec[0]	ByteRec[1]	ByteRec[2]	ByteRec[3]

Ainsi, si on reprend le principe de fonctionnement évoqué dans la section 4.2.3 :

`P4` à `P7` sont toujours à 1 grâce aux résistances de *pull-up*. Cependant, pour une raison non identifiée, il y a toujours un décalage de 1 bit dans `MISO` par rapport à ce qui est attendu. Ainsi, `MISO` correspond toujours à « `P6 P5 P4 P3 P2 P1 P0 1` », alors qu'il devrait correspondre à « `P7 P6 P5 P4 P3 P2 P1 P0` ». Le tableau ci-dessous tient donc compte de cette anomalie :

TABLEAU 5-15 – MISO ATTENDU EN FONCTION DE LA COLONNE

	Si aucun appui				Si appui col 1				Si appui col 2				Si appui col 3				Si appui col 4			
Broche	P0	P1	P2	P3	P0	P1	P2	P3	P0	P1	P2	P3	P0	P1	P2	P3	P0	P1	P2	P3
Valeur lue	1	1	1	1	1	1	1	0	1	1	0	1	1	0	1	1	0	1	1	1
MISO binaire	1111 1111				1110 1111				1111 0111				1111 1011				1111 1101			
MISO décimal	255				239				247				251				253			

La figure ci-dessous permet d'illustrer le tableau précédent :

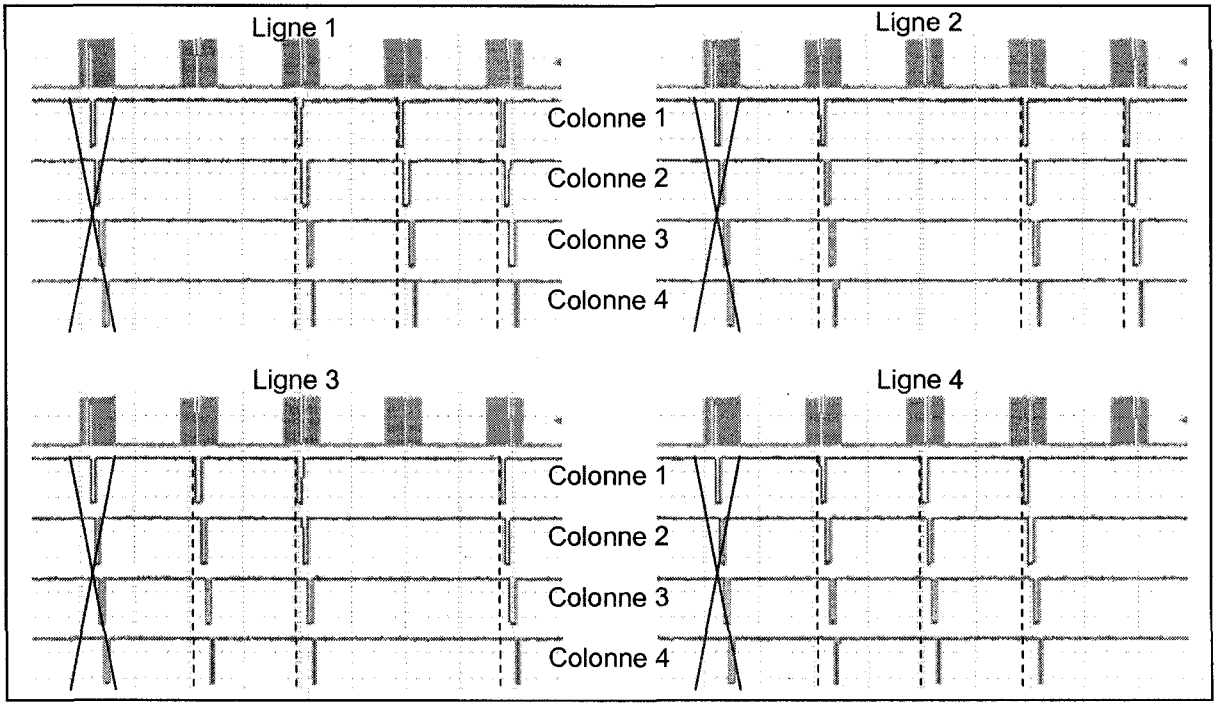


Figure 5-4 – Chronogrammes pour l'identification d'une touche

Ainsi, pour résumer :

TABLEAU 5-16 – OBTENTION DE L'ID D'UNE TOUCHE A PARTIR DE BYTEREC

Cycle (SCK)	Trame 1	Trame 2 = ByteRec[0]	Trame 3 = ByteRec[1]	Trame 4 = ByteRec[2]	Trame 5 = ByteRec[3]	ID touche lue
Lecture (MOSI)	(à ignorer)	255	239	239	239	1
		255	247	247	247	2
		255	251	251	251	3
		255	253	253	253	4
		239	255	239	239	5
		247	255	247	247	6
		251	255	251	251	7
		253	255	253	253	8
		239	239	255	239	9
		247	247	255	247	10
		251	251	255	251	11
		253	253	255	253	12
		239	239	239	255	13
		247	247	247	255	14
		251	251	251	255	15
		253	253	253	255	16

Du point de vue du code, on utilise une machine à états finis dont l'état change à chaque coup du Timer1. Le code de cette portion est disponible en Annexe, à la section 0.

Interprétation de la touche lue

L'interprétation de la touche obtenue se fait en 2 étapes :

Dans un premier temps, on utilise la fonction `InterpKeyKeyboard()` qui, à partir du tableau `ByteRec[]`, renvoie l'`ID` de la touche lue. Si aucune touche n'est reconnue, la fonction renvoie « 99 ». Si on détecte l'appui d'une touche mais que `ByteRec[]` ne correspond à aucune touche connue, la fonction renvoie « 0 ». Pour simplifier, cette fonction utilise le tableau Tableau 5-16 afin de trouver la correspondance entre `ByteRec[]` et l'`ID`.

Dans un deuxième temps, on utilise une portion de la fonction `ReadKeyboard()` afin de réaliser une fonction anti-rebonds logicielle. Pour cela, on stocke les ID renvoyées par la fonction `InterpKeyKeyboard()` dans des ID temporaires. Ainsi :

```
ID_old_old = ID_old;                    // Debounce : ID(n-2) = ID(n-1)
ID_old = ID;                            // Debounce : ID(n-1) = ID(n)
ID = InterpKeyKeyboard (ByteRec);       // Nouvel ID

if ((ID_old_old == 99) && (ID_old == ID) && (ID >0) && (ID <= NB_KEYS)) {
                                        // Is Key ID a newly pressed key ID?
     ...                                // Do necessary actions
}
```

Les conditions à remplir pour valider un appui touche sont les suivantes :

- Si la lecture d'il y a 2 cycles a montré qu'aucun appui n'était en cours ;
- Et si la lecture du cycle précédent ainsi que la lecture du cycle en cours montrent qu'une même touche a été appuyée ;
- Et si l'ID correspond à une touche connue (1..16).

Les actions effectuées en cas de détection d'une touche seront abordées dans la section Application (5.5).

Mise à jour des LEDs

Les explications qui suivent sont dans la continuité de ce qui a été exposé précédemment, dans la section Interfaçage des LEDs du paragraphe 4.2.3, en particulier le Tableau 4-6. Il est conseillé au lecteur de s'y référer, afin de faciliter la compréhension de ce qui suit.

Puisque les LEDs du clavier sont matricées, il n'est possible d'allumer qu'une seule LED à la fois. De ce fait, afin que l'utilisateur puisse voir un clavier illuminé (et non une seule touche), il faudra utiliser la propriété de persistance rétinienne[1] de l'œil humain et alterner les LEDs allumées très rapidement.

Ainsi, à chaque itération, le programme va parcourir la structure contenant toutes les informations à propos des touches, incluant la couleur de chaque touche et le drapeau indiquant si la LED est actuellement allumée ou pas.

Puis, on va envoyer les informations nécessaires à l'électronique permettant d'interfacer les LEDs du clavier, en l'occurrence les 2 composants 74HC4094 du circuit.

---

[1] Wikipedia – Persistance Rétinienne – http://fr.wikipedia.org/wiki/Persistance_r%C3%A9tinienne

Ces opérations devront être effectuées très rapidement. Dans notre cas, on changera de LED à allumer toutes les 500µs, ce qui correspond à la durée d'un *tick* (unité de temps de l'OS), afin que toutes les LEDs allumées du clavier puissent être parcourues avant que l'œil ne puisse voir de clignotement.

Pour cela, on fait appel à la fonction suivante toutes les 500µs :

```
void UpdateLEDs(void)
```

Cette fonction fonctionne avec un compteur `LEDsUpdateCnt`. Elle vérifie d'abord si une des 3 LEDs de la `LEDsUpdateCnt`ème touche du clavier doit être allumée. En effet, il est possible, à l'aide de certaines fonctions prévues et qui seront abordées ultérieurement, d'éteindre complètement ou d'allumer à nouveau le clavier.

Si aucune LED ne doit être allumée, alors la fonction envoie la trame correspondant (« `0x3F 0x3F` ») au port SPI. Si une des LEDs doit être allumée, alors la trame correspondant (voir le Tableau 4-6) est envoyée au port SPI.

Afin de connaitre la trame correspondant en utilisant des masques, on utilise la fonction :

```
INT16U TurnOnLED  (INT8U column, INT8U row, INT8U color)
```

Bien entendu, afin de synchroniser les échanges entre le port SPI du microcontrôleur LPC2106 et les 74HC4094, on utilise les 2 signaux de contrôle `CS2` et `CS3`. CS2 est connecté au 74HC4094 (U13), et CS3 au 74HC4094 (U13). Lorsque CS2 ou CS3 est à 1, cela permet de valider les sorties parallèles du composant concerné. Afin d'assurer le bon échange chronologique des signaux, on utilise la fonction de délai :

```
void DelayLED(int cpt)
```

Il peut être intéressant d'examiner les chronogrammes de la Figure 4-9 et de la Figure 4-10.

Par ailleurs, on dispose des fonctions suivantes :

```
void TurnOnLEDs(void)
void TurnOffLEDs (void)
```

Ces fonctions mettent à 1 (pour éteindre) ou à 0 (pour allumer) le drapeau `flagLEDsOff` afin de contrôler si l'ensemble des LEDs doit être allumé ou non. Ceci sera particulièrement

pratique lorsqu'on souhaitera éteindre les LEDs de l'écran après un certain délai d'inactivité de la part de l'usager sur le téléphone, et les rallumer au prochain évènement.

Tests

Afin de tester le bon fonctionnement du clavier, on teste dans un premier temps la détection de l'appui d'une touche. Pour cela, on affiche sur l'émulateur de terminal, via l'UART0, les touches détectées. A des fins de débogage, il est possible d'afficher les octets lus sur le port SPI, afin de s'assurer que ce qui est lu correspond bien à ce qui est attendu.

Pour vérifier le fonctionnement des LEDs du clavier, on peut dans un premier temps choisir d'allumer une seule LED avec une seule couleur. Puis, on peut choisir d'illuminer plusieurs touches du clavier, de différentes couleurs, afin de vérifier l'alternance des LEDs et l'absence de scintillement pour l'œil humain. Il s'agit là d'un test visuel subjectif.

5.4.3 Boîte à outils relative à l'afficheur

Initialisation

L'initialisation de l'afficheur nécessite les étapes suivantes :

TABLEAU 5-17 – ÉTAPES D'INITIALISATION DE L'AFFICHEUR

Étape	Fonction appelée
Initialisation de l'I²C	InitI2C()
Envoi de l'adresse du périphérique esclave (0x50 par défaut) (utile si plusieurs esclaves)	SendAddressI2C(0x50)
Initialisation des caractères spéciaux	InitAccentsDisplay()
Initialisation de l'écran d'accueil (facultatif)	SaveAsStartupDisplay()
Effacer l'écran	ClearDisplay()
Allumer l'écran si nécessaire	TurnOnDisplay()

Ces étapes sont regroupées dans la fonction :

```
void InitDisplay(void)
```

Afficher un caractère

Pour envoyer un caractère à afficher, il suffit, une fois les initialisations nécessaires effectuées, d'envoyer sur le bus I²C chaque caractère souhaité. Ainsi, on a créé les deux fonctions suivantes qui réalisent ces opérations :

```
void WriteChDisplay(unsigned char ch)
```

```
void WriteDisplay(unsigned char string[])
```

Envoi d'une commande

L'afficheur se commande à partir de trames I²C. Un résumé de toutes les commandes possibles est disponible à la section 11 "*Command Summary*" du manuel technique de l'afficheur [MO, 2006], à la page 41.

Ainsi, par exemple, pour envoyer la commande *Move Cursor Back*, il suffit d'envoyer les trames suivantes :

TABLEAU 5-18 – ENVOI DE LA COMMANDE « MOVE CURSOR BACK » A L'AFFICHEUR

Trames	1ère trame	2ème trame
en hexadécimal	0xFE	0x4C
en décimal	254	76
en ASCII	254	"L"

Afin d'assurer une grande modularité du système, on crée ainsi la fonction `CursorBackDisplay()` qui fera elle-même appel à la fonction `WriteI2C()` à 2 reprises, pour envoyer « `0xFE` » puis « `0x4C` » à l'afficheur.

De même, on construit les fonctions suivantes :

```
void TurnOnDisplay(void)
void TurnOffDisplay(void)
void ClearDisplay(void)
void NewLineDisplay(void)
void CursorBackDisplay(void)
void SaveAsStartupDisplay(void)
void SetCursorPosDisplay(int col, int row)
```

Cas des caractères spéciaux

L'afficheur n'étant pas prévu pour être utilisé pour la langue française, il est impossible d'écrire avec des caractères spéciaux tels que « é », « è », « à »... Cependant, l'utilisateur a la possibilité de créer ces caractères.

Pour commencer, il convient d'envoyer les commandes « `0xFE 0x4E [refID] [data]` » (en hexadécimal). `refID` est la référence du caractère spécial, et `data` contient le dessin de ce caractère.

Prenons l'exemple du caractère « é », pour lequel on attribuera l'ID « `1` ». La chaine « `data` » est la suivante :

TABLEAU 5-19 – CONSTRUCTION DU CARACTERE SPECIAL « E » POUR L'AFFICHEUR

Construction du caractère « é »					Caractères hexadécimaux à envoyer
0	0	0	1	0	0x02
0	0	1	0	0	0x04
0	1	1	1	0	0x0E
1	0	0	0	1	0x11
1	1	1	1	1	0x1F
1	0	0	0	0	0x10
0	1	1	1	0	0x0E

Ainsi, les commandes I²C à envoyer pour initialiser le caractère « é » seront :

« `0xFE 0x4E 0x01 0x02 0x04 0x0E 0x11 0x1F 0x10 0x0E` ».

Il en va de même pour les autres caractères spéciaux. A noter qu'il est déconseillé d'initialiser un caractère avec l'ID « `0` » : le caractère ainsi initialisé s'affichera en position (1,1) de façon aléatoire et involontaire. Il est possible d'initialiser 7 caractères (1-7). Une fois tous les caractères souhaités initialisés, il convient de charger ces caractères, à l'aide de la séquence suivante : « `0xFE 0xC0 0x00` ».

Pour réaliser cette opération d'initialisation, il existe la fonction suivante :

```
void InitAccentsDisplay(void)
```

L'utilisation de ces caractères spéciaux est la suivante : à chaque caractère que l'on souhaite envoyer, une fonction dédiée regarde si ce caractère fait partie des caractères spéciaux définis précédemment (« é », « è », « à »), et si c'est le cas, remplace ce caractère par l'ID correspondant. Ceci est réalisé par la fonction :

```
unsigned char ConvertAccentsDisplay(unsigned char Data)
```

Tests

Pour tester l'afficheur, il suffit d'envoyer un certain nombre de commandes sur le bus I²C (envoyer un caractère, déplacer le curseur, effacer le dernier caractère, ...) et de vérifier que l'opération s'est bien effectuée sur l'écran.

5.5 Développement de l'application haut niveau

5.5.1 Architecture

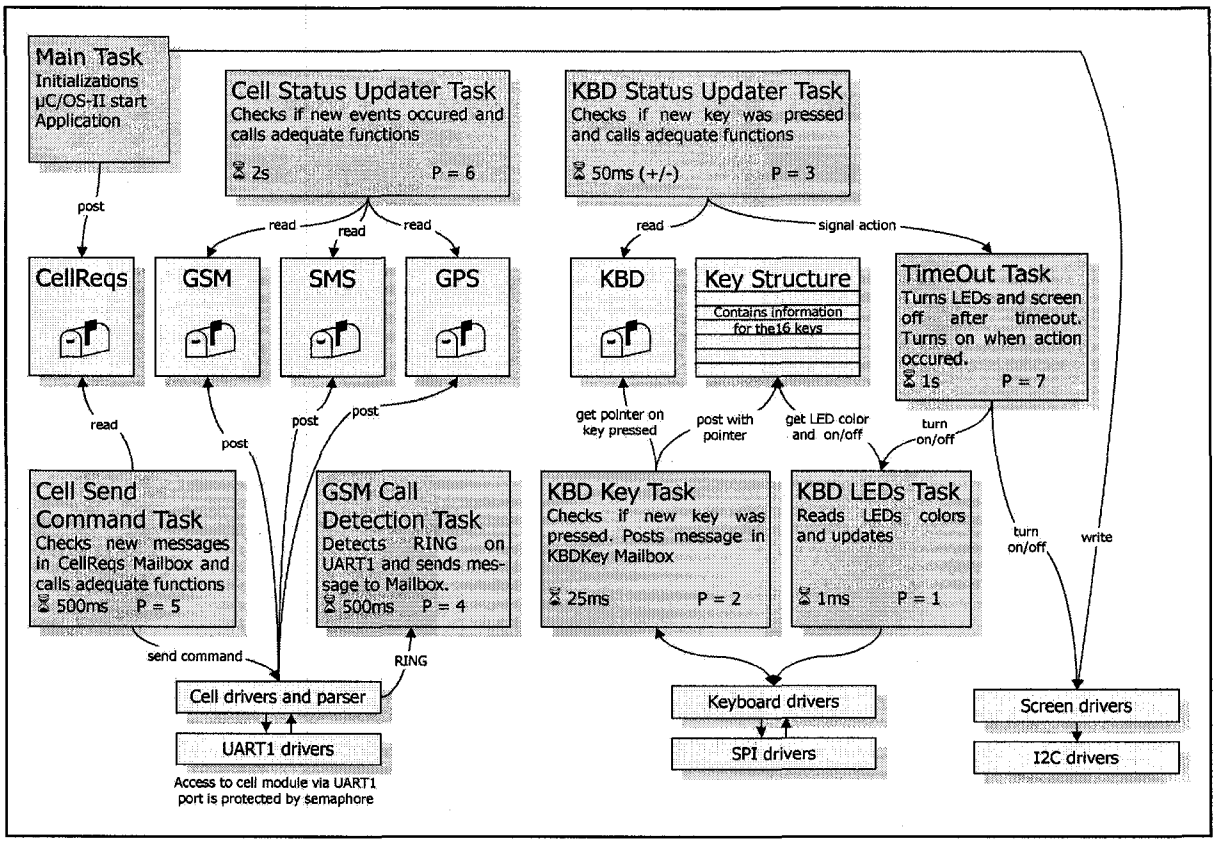


Figure 5-5 – Architecture globale de l'application

La figure ci-dessus illustre la mise en œuvre logicielle de ce projet. En effet, on peut voir l'organisation en tâches de l'application, ainsi que les boites à outils et les pilotes bas-niveau. Les tâches communiquent entre elles par le biais de boites aux lettres. Tout ceci sera détaillé au cours de ce chapitre.

5.5.2 Cas d'utilisation

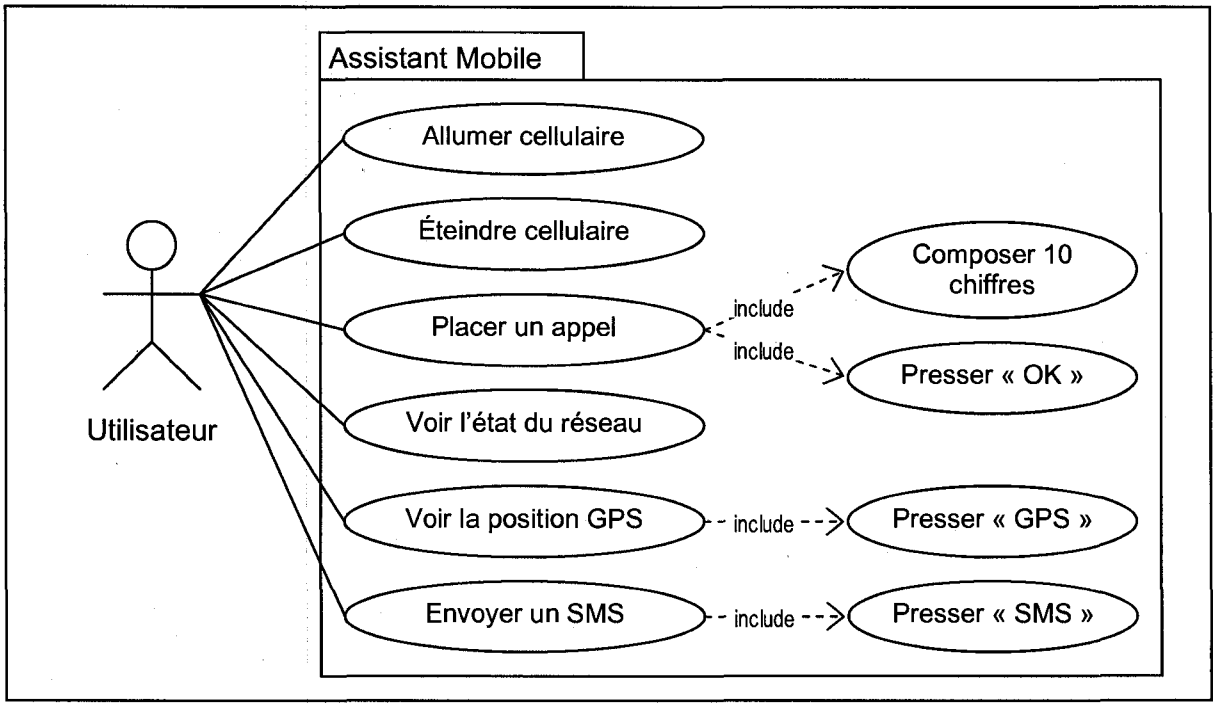


Figure 5-6 – Diagramme des cas d'utilisation de l'assistant mobile

5.5.3 Composants

Afin de réaliser l'application finale qui va régir tous les composants du système, on utilise les possibilités offertes par µC/OS-II [LABROSSE, 2002] : la gestion de tâches concurrentes (*multitasking*), les boites-aux-lettres (*mailboxes*), et les sémaphores (*semaphores*).

Tâches

On identifie les différentes tâches nécessaires :

TABLEAU 5-20 – TACHES CREEES POUR L'APPLICATION

Tâche	Description	Priorité	Périodicité
KBD LED Task	Réalise la mise-à-jour de l'allumage des LEDs du clavier à partir des informations de couleur contenues dans la structure de touches	1	1ms
KBD Key Task	Vérifie si une nouvelle touche du clavier a été pressée. Si c'est le cas, la tâche cherche le pointeur sur cette touche dans une structure de touches, et poste un message avec ce pointeur	2	25ms
Keyboard Status Updater Task	Vérifie si un nouvel appui touche s'est produit au niveau du clavier. Si c'est le cas, la tâche effectue les actions nécessaires en conséquence	3	50ms
GSM Call Detection Task	Détecte l'arrivée du message non sollicité « RING » provenant du cellulaire, ce qui permet de détecter un appel entrant. Si c'est le cas, poste un message afin de prévenir l'application	4	500ms
Cell Send Command Task	Vérifie l'arrivée de nouvelles commandes à envoyer au cellulaire de la part de l'application. Si c'est le cas, la tâche appelle les fonctions nécessaires à l'envoi de ces commandes	5	500ms
Cell Status Updater Task	Vérifie si un nouvel évènement ou une réponse à une commande s'est produite, au niveau du cellulaire. Si c'est le cas, la tâche effectue les actions nécessaires en conséquence	6	2s
TimeOut Task	Éteint les LEDs du clavier et l'écran après une période d'inactivité. Dès qu'un évènement (« RING » ou appui touche) se produit, rallume les LEDs du clavier et l'écran	7	1s
MainAppTask	Exécute les actions nécessaires au fonctionnement général	10	500ms

Boites aux lettres

On identifie les différentes boites-aux-lettres nécessaires :

TABLEAU 5-21 – BOITES AUX LETTRES CREEES POUR L'APPLICATION

Boite aux lettres	Description	Postée par	Lue par
CellReqsMbox	Contient les commandes à envoyer au module cellulaire. La nature de la commande est identifiée par un numéro	Application ou KBD Status Updater Task	Cell Send Command Task
GSMMbox	Contient les réponses aux commandes envoyées concernant la fonctionnalité GSM du module cellulaire. La nature de la réponse est identifiée par un numéro	Fonctions de la boîte à outils du cellulaire	Cell Status Updater Task
SMSMbox	Contient les réponses aux commandes envoyées concernant la fonctionnalité SMS du module cellulaire. La nature de la réponse est identifiée par un numéro		
GPSMbox	Contient les réponses aux commandes envoyées concernant la fonctionnalité GPS du module cellulaire. La nature de la réponse est identifiée par un numéro		
KBDMbox	Contient le dernier appui touche détecté sur le clavier. La touche pressée est identifiée grâce à un pointeur sur la structure de touches	KBD Key Task	KBD Status Updater Task

Sémaphores

On identifie les différents sémaphores nécessaires :

TABLEAU 5-22 – SEMAPHORES CREES POUR L'APPLICATION

Sémaphore	Description	Pris par	Libéré par
AccessToCell Sem	Permet de réguler l'accès au module cellulaire. En effet, il est nécessaire d'attendre qu'une commande soit envoyée complètement avant d'en envoyer une autre	Fonctions de la boîte à outils du cellulaire	Fonctions de la boîte à outils du cellulaire
Timeout Sem	Permet la comptabilisation facile du temps pour la tâche TimeOut Task, grâce à la possibilité de limiter le temps d'attente d'un sémaphore	TimeOut Task	Keyboard Status Updater Task

5.5.4 Fonctionnalités

Plus le chiffre correspondant à la priorité d'une tâche est faible, plus cette tâche est prioritaire. Ainsi, en règle générale, plus une tâche doit être effectuée souvent, plus on lui assigne une priorité élevée (et donc un chiffre faible). C'est d'autant plus vrai que dans notre cas, il faut scanner les touches du clavier (à l'aide de KBD Key Task) à intervalles très réguliers afin de ne pas rater la détection d'un appui de touche. Cette tâche ne peut souffrir de beaucoup de retard.

Afin de démontrer la viabilité de l'architecture réalisée, quelques opérations de base qui sont à réaliser par le téléphone seront prises afin de réaliser un exemple. Les schémas représentés à la Figure 5-7, à la Figure 5-8 et à la Figure 5-9 ne prennent pas en compte tous les détails du code livré, mais permettent une bonne compréhension de l'exploitation des boîtes à outils et des fonctionnalités offertes par µC/OS-II.

Placer un appel

Voici Figure 5-7 les différentes étapes à réaliser.

Envoyer un SMS

Voici Figure 5-8 les différentes étapes à réaliser.

Lire la position GPS

Voici Figure 5-9 les différentes étapes à réaliser.

D'autres fonctions peuvent bien entendu être effectuées, celles mentionnées ci-dessus sont pour le moment des exemples.

5.5.5 Tests

Il est possible de tester les différentes fonctionnalités en vérifiant les communications entre les différents modules, notamment en écrivant sur l'émulateur de terminal un texte indiquant que tel message de telle boite aux lettres vient d'être lu. Cela permet par exemple de

détecter que l'application a bien été informée de l'appui d'une touche dans un délai raisonnable.

Les 3 figures ci-dessous mettent en évidence les différentes étapes nécessaires pour pouvoir placer un appel, envoyer un SMS et obtenir la position GPS, effectuées par les différentes tâches.

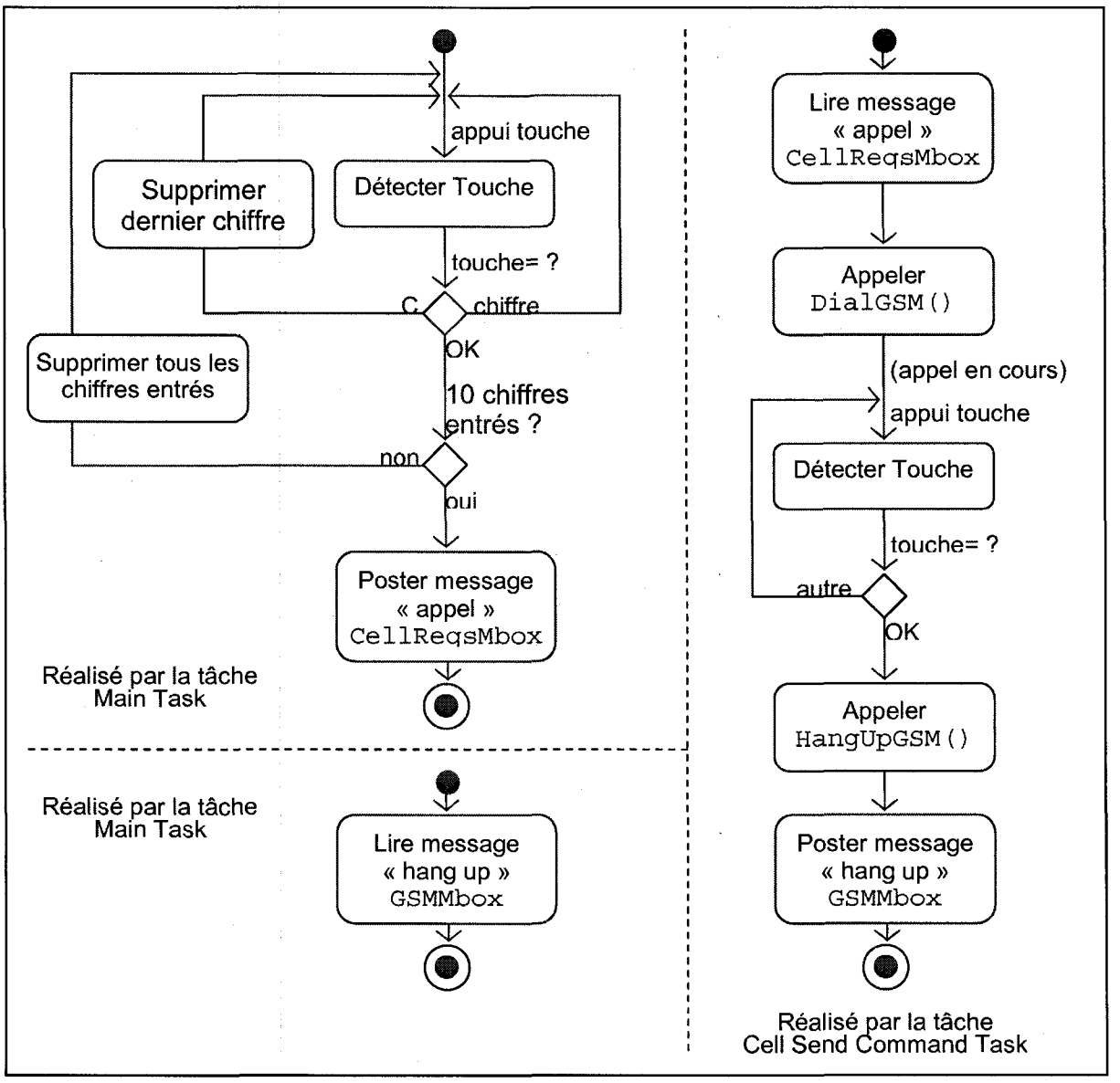


Figure 5-7 – Diagramme d'activité « placer un appel »

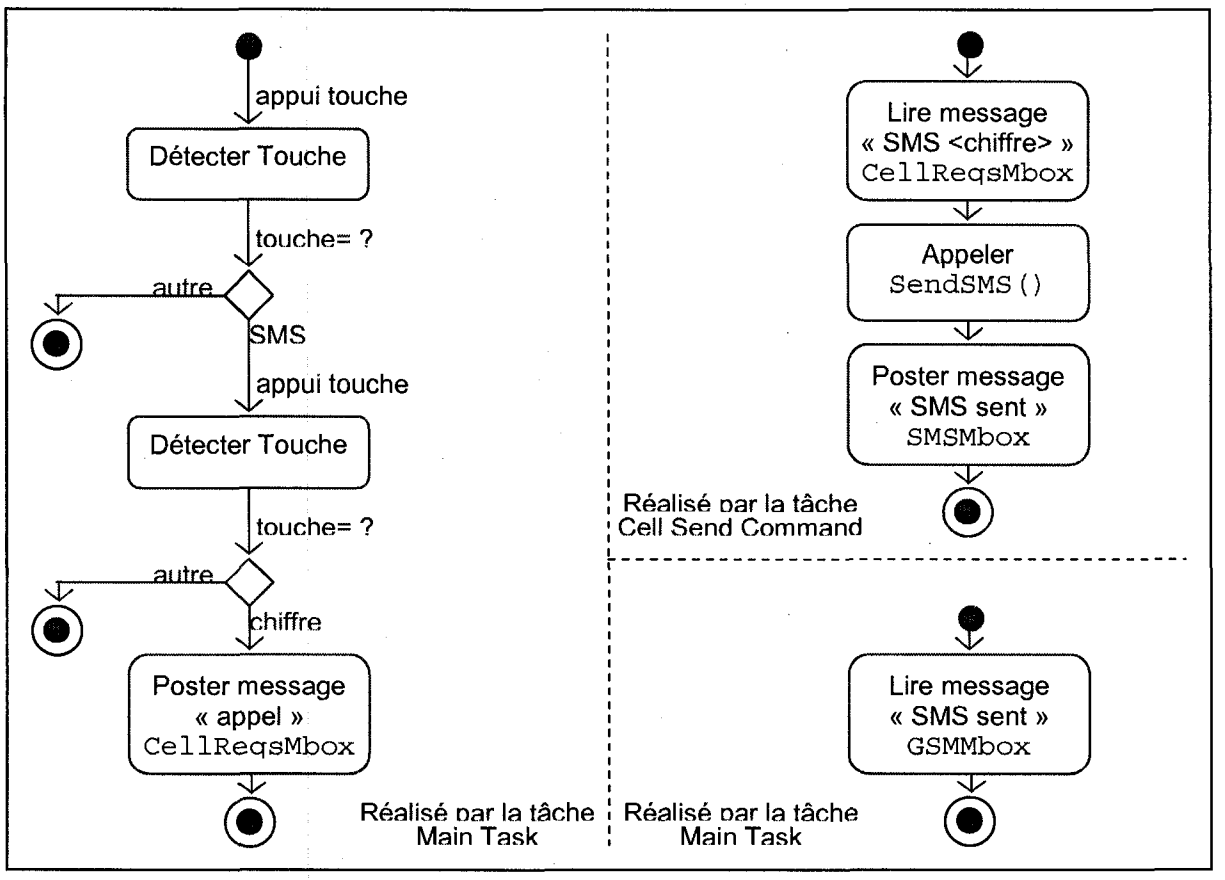


Figure 5-8 – Diagramme d'activité « envoyer un SMS »

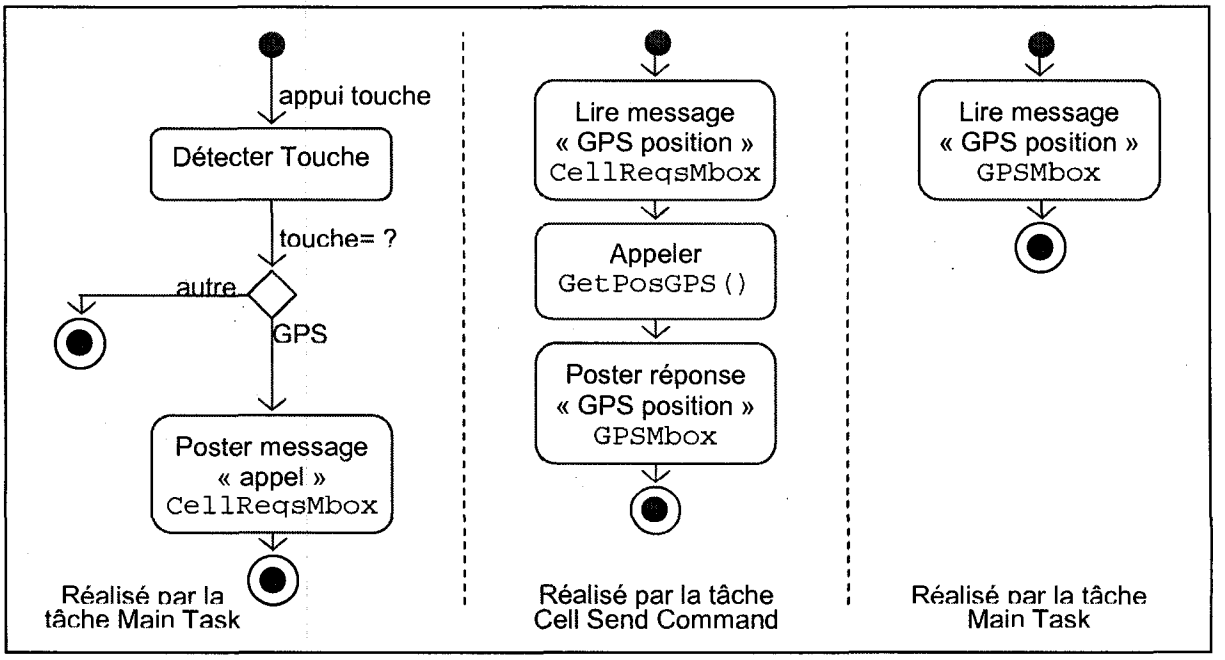


Figure 5-9 – Diagramme d'activité « obtenir la position GPS »

CHAPITRE 6.

APPLICATION VALIDANT LA MISE EN ŒUVRE

6.1 Retour sur les besoins

A la section 2.3 - Besoins recensés, figure un certain nombre de besoins auxquels ce projet doit répondre. Il est ainsi aisé de constater que la plateforme – matérielle comme logicielle – est en adéquation avec les besoins énoncés initialement. En voici un résumé pour rappel :

6.1.1 Sur le plan matériel

Les besoins étaient les suivants :

- une plateforme cellulaire de dimensions et poids raisonnables, pouvant se fixer sur un support quelconque ;
- un module cellulaire permettant de prendre en charge les communications avec le réseau cellulaire ainsi que la géolocalisation ;
- un écran assez lumineux ;
- un clavier dont les boutons sont suffisamment gros et espacés ;
- un haut-parleur permettant une sortie sonore avec un volume suffisamment fort ;
- une batterie permettant d'avoir un système autonome sur le plan énergétique.

La description du matériel dans la section 3.2 Composants matériels permet de voir que les éléments matériels choisis correspondent aux besoins évoqués. C'est d'ailleurs à partir du matériel choisi adapté que tout le reste du projet a été élaboré.

Les précautions qui ont été employées dans le développement matériel, notamment en termes de disponibilité des broches d'entrée-sortie, permettront sans aucun doute l'ajout d'autres périphériques sur le microcontrôleur.

6.1.2 Sur le plan logiciel

Les besoins étaient les suivants :

- pouvoir placer un appel, répondre à un appel et terminer une communication ;
- pouvoir envoyer un SMS d'urgence à un numéro préprogrammé contenant la position GPS de l'usager, avec le choix d'un message parmi plusieurs ;
- ... prévoir des extensions à ce qui sera déjà implémenté.

Le logiciel dans l'état actuel permet en effet ces fonctionnalités de base. Cependant, tout au long du développement logiciel du projet, l'accent a été mis sur l'importance de son extensibilité. C'est d'ailleurs la raison pour laquelle le choix a été porté sur l'OS µC/OS-II.

6.2 Extensions possibles

6.2.1 Extensions matérielles

Amélioration du contrôle de l'alimentation

On peut gérer l'alimentation de la plateforme de façon plus intelligente, en utilisant la broche PWRMON du module cellulaire qui permet d'avoir une « image » de l'état éteint / allumé du module cellulaire. En effet, il suffirait d'allumer le module cellulaire à l'aide du bouton poussoir relié à la broche ON/OFF, et de « réveiller » (sortir de veille) le microcontrôleur avec la broche PWRMON reliée à une broche d'interruption externe. Ainsi, si le module cellulaire est éteint, le microcontrôleur et ses périphériques sont en veille. Si le module cellulaire est allumé, alors l'ensemble est opérationnel.

Cela permettrait une meilleure gestion de l'alimentation, puisque quand le cellulaire est éteint et que le reste est en veille, la consommation électrique de l'ensemble est très minime.

Ajout de nouveaux périphériques

Par exemple, imaginons qu'un développeur souhaite ajouter une nouvelle interface permettant de commander le téléphone par un simple contacteur. Il suffirait alors de

développer les pilotes bas-niveau, puis les fonctions appelées dans ce document « boîtes-à-outils » utilisant ces pilotes. Enfin, l'intégration haut-niveau pourra probablement se faire en créant une nouvelle tâche, à exécution périodique, dans le programme principal. La communication avec les autres sections du programme pourra se faire par boîtes-aux-lettres par exemple.

Ajout d'une alternative à la technologie GPS

Le signal GPS ne pénétrant pas les bâtiments, l'ajout d'un composant permettant de poursuivre la localisation tel qu'un MEMS ou autre micro centrale inertielle permettrait de compenser cette absence temporaire de signaux. On peut également imaginer utiliser une autre technologie de localisation restreinte à un bâtiment, comme dans un centre d'achat, mais nécessitant malheureusement l'installation d'une infrastructure coûteuse.

Ajout de boutons facilitant la navigation

Il pourrait être également intéressant, dans une version future, de disposer des boutons de part et d'autre de l'écran afin de pouvoir, à l'aide du texte affiché à l'écran, guider l'utilisateur, à l'image de certains distributeurs automatiques de billets de banque.

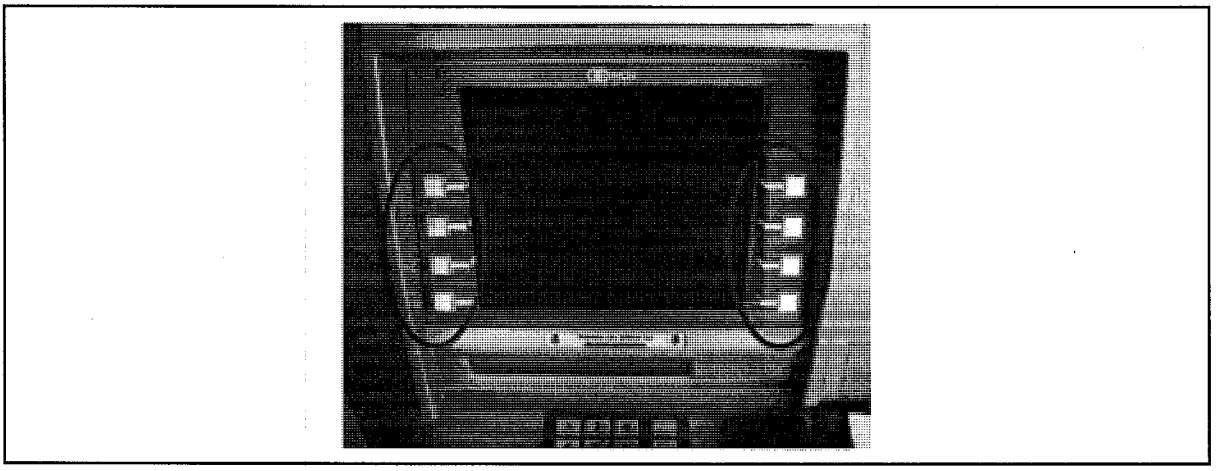


Figure 6-1 – Boutons d'un distributeur automatique de billets de banque

6.2.2 Extensions logicielles

Pour le moment, sont seulement implémentées les fonctions mentionnées dans la section 6.1.2. Cependant, on peut imaginer de très nombreuses applications à la plateforme. Pour cela, on dispose de la possibilité de connaitre les coordonnées GPS, ainsi que d'une connexion GPRS (*data*) fournie par le module cellulaire. Ainsi, on va pouvoir utiliser, par exemple, soit une *API Google Maps*[1], *Microsoft MapPoint Web Service*[2], ou tout autre serveur similaire, soit un serveur « maison » de type *Apache Tomcat*[3]. Voici un diagramme de cas d'utilisation possible :

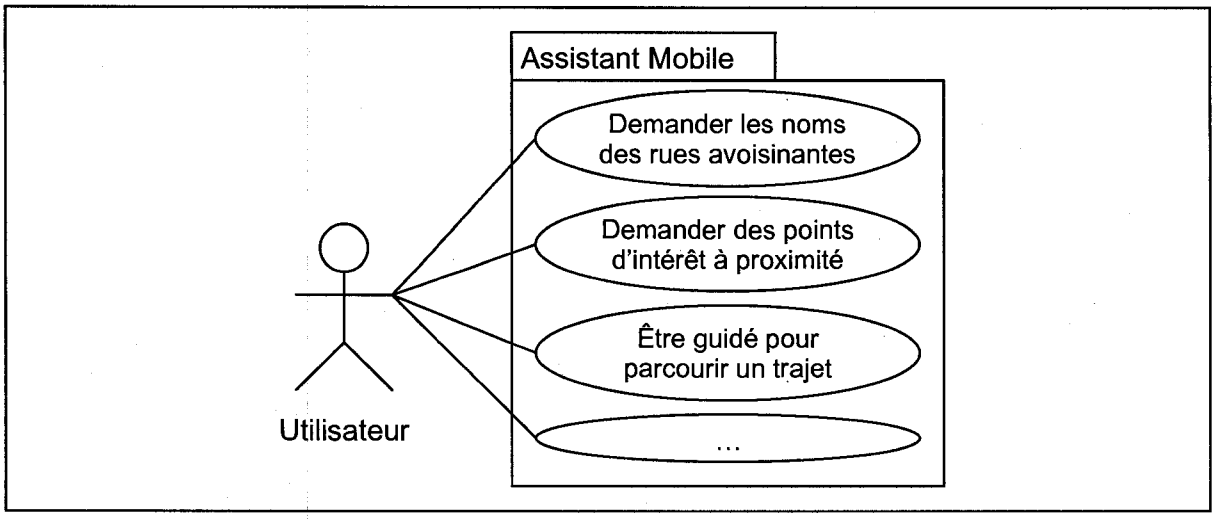


Figure 6-2 – Cas d'utilisation des services web de géolocalisation

Les architectures pourraient être une des suivantes :

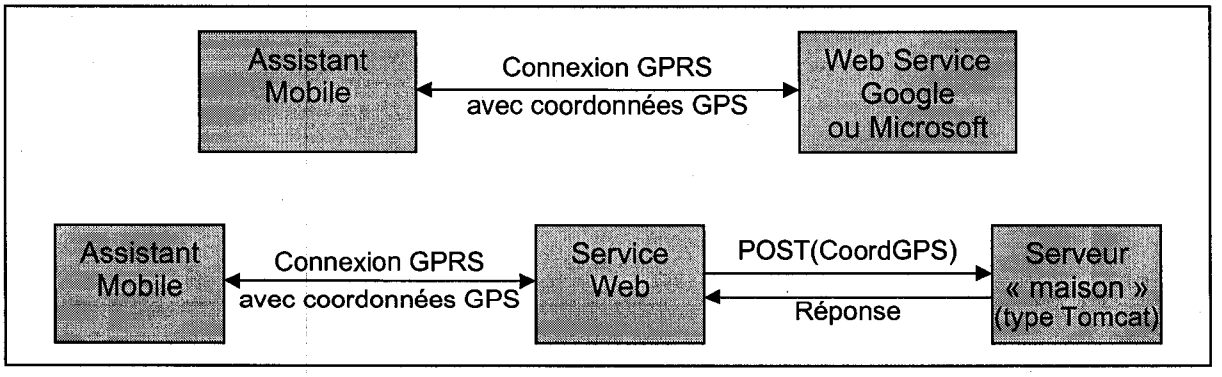


Figure 6-3 – Architectures possibles pour des services de géolocalisation

[1] Google Maps – http://maps.google.ca
[2] Microsoft MapPoint Web Service – http://www.microsoft.com/mappoint/products/webservice/default.mspx
[3] Apache Tomcat – http://tomcat.apache.org/

Un article suggère ce besoin :

> "*Once a device is accessible to a user with visual impairments, a world of possibilities is enabled. [...] Personal location-based services would also be a standard part of the package. These services would use the capability of the new phones to provide the user's current location, as Global Positioning System receivers now do. These services will make information about the user's current position ("you are at the corner of First and Main"), as well as directions based on either walking or public transportation, available. In addition to position and navigational services, information will be readily available for points of interest, such as restaurants and emergency services, and would be able to pinpoint the location of a mobile user.*", [FRUCHTERMAN, 2003]

Connaitre le nom des rues environnantes

Par exemple, certaines personnes ont de la difficulté à lire les panneaux sur lesquels figurent les noms de rues. Pour les voir, il leur faut s'en approcher, ce qui peut être périlleux compte tenu de l'emplacement de certains de ces panneaux. On peut alors imaginer récupérer le nom des rues avoisinantes à l'aide d'un service similaire à *Microsoft MapPoint Web Service* et indiquer à l'utilisateur le nom de la rue dans laquelle il se trouve.

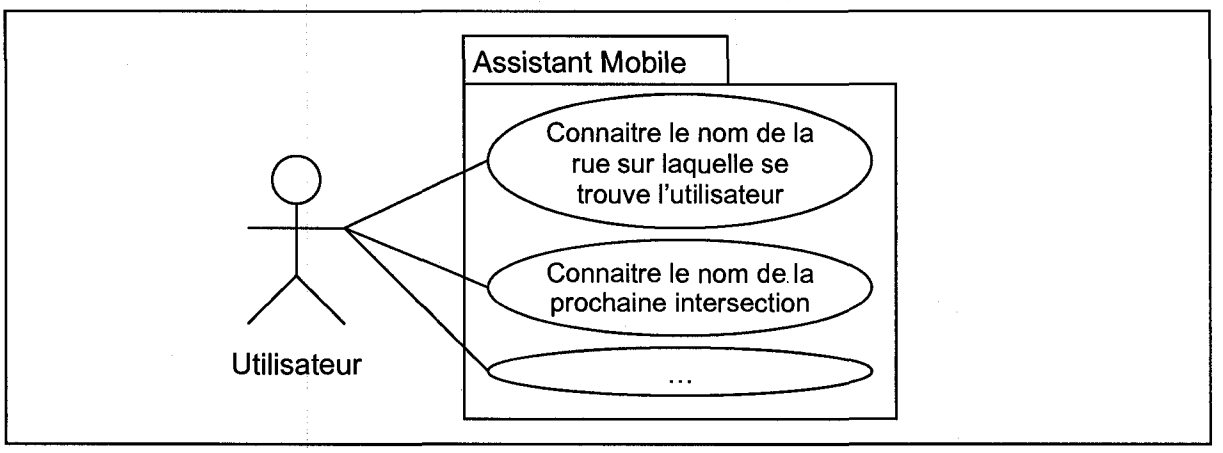

Assistant Mobile
Connaitre le nom de la rue sur laquelle se trouve l'utilisateur
Connaitre le nom de la prochaine intersection
...
Utilisateur

Figure 6-4 – Cas d'utilisation pour connaitre le nom des rues avoisinantes

Effectuer un appel d'urgence vers un numéro préprogrammé

Un usager en détresse n'a pas nécessairement la possibilité de placer un appel conventionnel d'urgence, dans des situations induisant un fort stress. Il serait donc intéressant de proposer à l'usager la possibilité d'appeler d'urgence un numéro de téléphone préprogrammé sur l'appui (ou l'appui répétitif) d'une simple touche. En plus de l'appel, on peut imaginer soit envoyer un SMS au destinataire de l'appel d'urgence, ou soit envoyer un courriel, l'informant de la position géographique (coordonnées GPS) de l'usager.

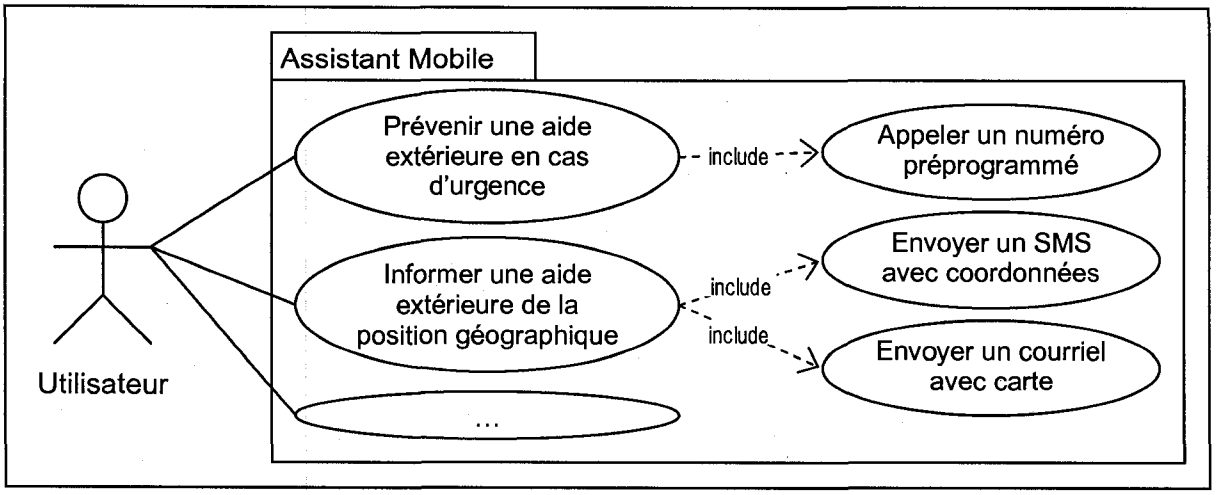


Figure 6-5 – Cas d'utilisation pour effectuer un appel d'urgence

Connaitre les instructions pour parcourir un trajet

Avec un récepteur GPS et une liaison vers un serveur comme celui de Google ou Microsoft, il est possible d'obtenir toutes les informations nécessaires au guidage de l'usager pour un trajet d'un point A à un point B, identifié par une adresse civique ou des coordonnées GPS. Cela est particulièrement pratique pour des personnes dont la mobilité est réduite, car dans la mesure où les déplacements sont pénibles, il est intéressant de les optimiser en limitant la distance parcourue, par exemple.

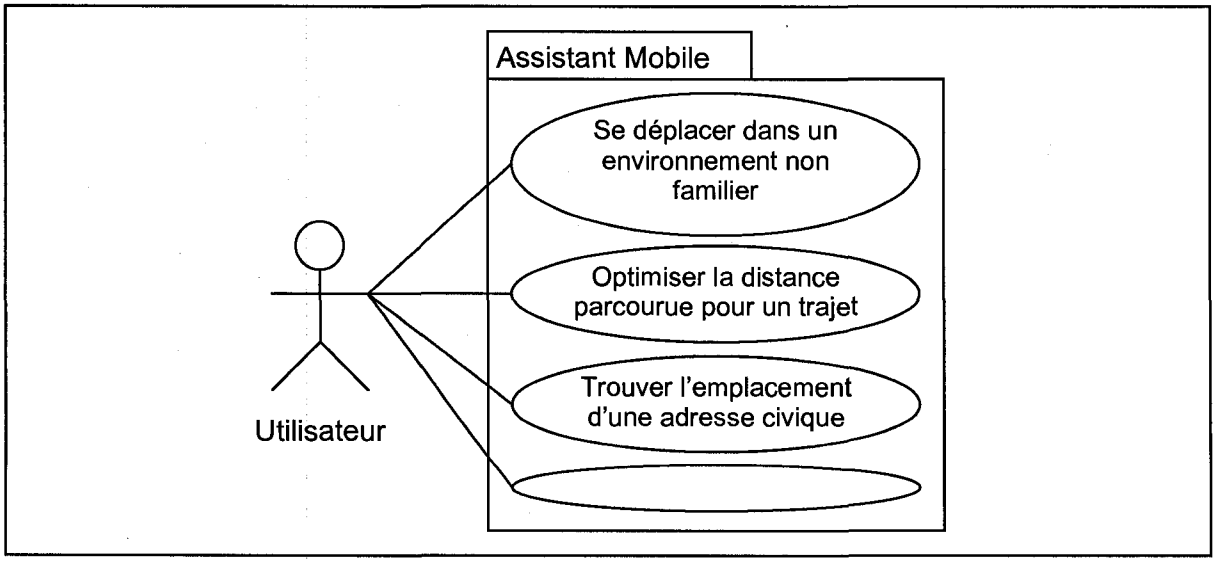

Assistant Mobile
Se déplacer dans un environnement non familier
Optimiser la distance parcourue pour un trajet
Trouver l'emplacement d'une adresse civique
Utilisateur

Figure 6-6 – Cas d'utilisation pour connaitre les instructions pour parcourir un trajet

Autres améliorations possibles

Il serait intéressant d'associer ou de compléter la localisation GPS avec la localisation par triangulation des cellules GSM. En effet, une cellule GSM étant de taille relativement restreinte (surtout en zone urbaine), il est possible, par recoupement des informations provenant de plusieurs cellules contigües, de connaitre la position d'un utilisateur de façon assez précise.

Il serait également souhaitable de disposer d'une vraie interface avec l'utilisateur, comprenant l'accès à un répertoire téléphonique, la rédaction de SMS, des fonctionnalités de réveil / alarme, etc... Tout ceci est possible de façon assez simple avec le module cellulaire. Il est par ailleurs possible d'utiliser l'interpréteur Python et les entrées-sorties du module cellulaire afin de faciliter les échanges avec le microcontrôleur.

Un guidage à l'aide des LEDs du clavier pourrait aider l'utilisateur dans le choix des touches. Par exemple, si on souhaite que celui-ci valide ou annule un choix, on pourrait allumer la touche « OK » en vert et allumer la touche « C » en rouge. Le code quant à l'allumage des LEDs est d'ailleurs déjà existant.

Avec la plateforme dans son état actuel, les possibilités sont multiples. En effet, le code actuel utilise moins de 45 ko sur les 128 ko disponibles [NXP, 2006ds]. Il reste donc 65% de la mémoire flash libre. La complexité du code contenu peut donc encore beaucoup évoluer.

CONCLUSION

Bilan

Ce projet consistait à développer une plateforme expérimentale présentant des fonctionnalités de communications GSM et GPS afin de faciliter l'accès aux technologies mobiles pour les personnes présentant des déficiences. Cette plateforme devait pouvoir effectuer les opérations de base de téléphonie cellulaire et de géolocalisation. Elle devait permettre d'être utilisée par des personnes à la vision, à l'audition ou à la dextérité limitée. Enfin, cette plateforme devait être ouverte à de futures améliorations, tant matérielles que logicielles.

Ainsi, après avoir réalisé une étude des publications parues autour de ce sujet, les composants matériels ont été choisis en conséquence. S'en est suivie une étape de découverte du matériel et de l'environnement logiciel associé. Puis, par incréments successifs, les périphériques matériels ont pu être incorporés au cœur du projet. Finalement, une dernière itération a consisté à réécrire le code logiciel afin de pouvoir profiter des avantages de µC/OS-II.

Le produit obtenu a la fin de ces étapes est conforme aux spécifications mentionnées notamment à la section 2.3 Besoins recensés : fonctionnel, il dispose de touches suffisamment grandes pour être facilement manipulable, d'un écran très brillant donc très visible, de tout le nécessaire pour permettre la géolocalisation, d'un haut-parleur puissant, et d'une batterie. Il permet d'appeler, de recevoir un appel, d'envoyer un SMS, et de recevoir la position GPS. Il constitue ainsi une bonne plateforme de départ, tant matérielle que logicielle, pour ajouter par la suite de nouveaux périphériques et de nouvelles fonctionnalités.

Perspectives

Cependant, cette plateforme est loin d'être un produit fini et commercialisable. Il faudra améliorer son ergonomie, développer un certain nombre de fonctionnalités liées au téléphone cellulaire ainsi qu'un système de géolocalisation plus poussé. En effet, pouvoir localiser et orienter une personne grâce aux possibilités du GPS apporterait un réel soutien aux personnes qui en ont le plus besoin. C'est ce qui fait de ce projet un assistant mobile et pas uniquement un téléphone cellulaire conventionnel.

D'autre part, cette plateforme est prévue pour pouvoir être adaptée à un large spectre de déficiences. En effet, bien qu'il soit souhaitable que cet appareil soit accessible au plus grand nombre, il existe une telle variété de besoins qu'il faudra une variété tout aussi importante de solutions à apporter pour que la plateforme réponde réellement aux besoins de chacun. A terme, un produit fini rendant service à ceux qui n'auraient pas pu avoir accès à la technologie cellulaire et à la géolocalisation autrement pourra voir le jour.

ANNEXES

A.1 Prototypes des fonctions principales

Le code et les commentaires étant en anglais, on conservera cette langue pour les prototypes des fonctions.

A.1.1 Pilotes bas-niveau

UARTs

Figurent ici uniquement les fonctions pour l'UART0, mais celles de l'UART1 sont identiques.

`void uart0Init(INT16U baud, INT8U mode, INT8U fmode)`		
Description	This function initializes the UART for asynchronous mode	
Arguments	baud mode fmode	baudrate (use macro in `uart.h`) number of bytes, parity, number of stop bits (see modes in `uart.h`) FIFO mode (see fmodes in `uart.h`)
Retour	`void`	

`int uart0Putch(int ch)`		
Description	This function puts a character into the UART output queue for transmission	
Arguments	`ch`	character to be transmitted
Retour	`ch`	`ch` on success, `-1` on error (queue full)

`const char *uart0Puts(const char *string)`		
Description	This function writes a NULL terminated 'string' to the UART output queue, returning a pointer to the next character to be written.	
Arguments	`string`	address of the string
Retour	`string`	a pointer to the next character to be written (\0 if full string is written)

`int uart0Getch(void)`		
Description	This function gets a character from the UART receive queue	
Arguments	`void`	
Retour	`ch`	character on success, `-1` if no character is available

`void uart0ISR(void)`		
Description	This function implements the ISR for UART0.	
Arguments	`void`	
Retour	`void`	

Bus SPI

`void InitSPI (void)`		
Description	SPI initializations	
Arguments	`void`	
Retour	`void`	

`void ReadWriteSPI (unsigned int Byte2Send)`		
Description	First part of a SPI Write/read cycle. Loading data to SPDR register starts clock and transfer	
Arguments	`Byte2Send`	Data to be sent by SPI port
Retour	`void`	

`unsigned int ReadWriteSPI_fin (void)`		
Description	Second (and last) part of a SPI Write/read cycle. Waits till transfer is done, checks SPI status register and reads received data	
Arguments	`void`	
Retour	`ByteRec`	Data received by SPI port

Bus I²C

`void InitI2C(void)`		
Description	I²C initializations	
Arguments	`void`	
Retour	`void`	

`void SendAddressI2C(unsigned char Addr)`		
Description	Sends the I²C Slave Address to be used	
Arguments	`Addr`	Slave address
Retour	`void`	

`unsigned char ReadI2C(void)`		
Description	Receives data from I²C port	
Arguments	`void`	
Retour	`I2C_I2DAT`	Data received from I²C port

`void WriteI2C(unsigned char Data)`		
Description	Sends data to I²C port	
Arguments	`Data`	Data to be sent to I²C port
Retour	`void`	

`void WriteStringI2C(char string[])`		
Description	Sends a string of characters to I²C port	
Arguments	`string`	String of characters to be sent to I²C port
Retour	`void`	

void StopI2C(void)		
Description	Stops the I2C communication	
Arguments	void	
Retour	void	

A.1.2 Boîtes à outils intermédiaires

Cellulaire

void InitCell(void)		
Description	Inits the cell phone device	
Arguments	void	
Retour	void	

void TurnOffCell(void)		
Description	Checks if Cell is ON and turns it OFF if necessary	
Arguments	void	
Retour	void	

INT8U GetPwrMonCell(void)		
Description	Checks if Cell is ON or OFF	
Arguments	void	
Retour	INT8U	1: Cell is ON - 0: Cell is OFF

void InitConfigCell(void)		
Description	Sends all necessary configuration commands to the cell	
Arguments	void	
Retour	void	

void DialGSM (unsigned char *pPhoneNb)		
Description	Dials the specified phone number and posts it in mailbox	
Arguments	*pPhoneNb	Pointer to phone number to be dialed
Retour	void	

void HangUpGSM(void)		
Description	Hangs up the current phone conversation and posts it in mailbox	
Arguments	void	
Retour	void	

void AnswerGSM(void)		
Description	Answers an incoming phone call and posts it in mailbox	
Arguments	void	
Retour	void	

INT8U CheckIncomingCallGSM(void)		
Description	Checks if incoming calls are waiting	
Arguments	void	
Retour	INT8U	1: A call is waiting - 0: No call waiting

void CheckNetworkStatusGSM(void)		
Description	Checks network status and updates variable NumCREG level (0-5), -1 if error	
Arguments	void	
Retour	void	

void TurnOnGPS(void)		
Description	Turns the GPS module on if module is off and posts it in mailbox	
Arguments	void	
Retour	void	

void TurnOffGPS(void)		
Description	Turns the GPS module off if module is on and posts it in mailbox	
Arguments	void	
Retour	void	

void GetPosGPS(void)		
Description	Gets the current GPS position and posts it in mailbox	
Arguments	void	
Retour	void	

void SendSMS(INT8U SmsID)		
Description	Sends the specified pretyped SMS to the specified phone number	
Arguments	SmsID	ID of the specified SMS
Retour	void	

void askGM862 (unsigned char * request,unsigned char * response)		
Description	Sends AT command to GM862 and waits for response	
Arguments	request response	The AT command sent to the GM862 module The response received from the GM862 module
Retour	void	

Clavier

Fonctions concernant le clavier en général :

void InitKeyboard(void)		
Description	Inits the keyboard	
Arguments	void	
Retour	void	

void InitKeyStruct (void)		
Description	Inits the Key Structure containing all Key data	
Arguments	void	
Retour	void	

`void AssignKeyStruct (INT8U index,INT8U kID,INT8U kColumn,INT8U kRow, INT8U kColor,INT8U kLedOn,unsigned char kSymbol)`		
Description	Adds a new key in the Key structure	

Arguments	`index`
	`kID`
	`kColumn`
	`kRow`
	`kColor`
	`kLedOn`
	`kSymbol`
		Index of the corresponding key in the structure
		ID of the Key to be stored
		Column of the Key to be stored
		Row of the Key to be stored
		Color of the Key to be stored
		Is the LED On? (1:yes - 0:no)
		Character represented on Key

Retour	`void`	

Fonctions concernant le fonctionnement des touches du clavier :

`void ReadKeyboard(void)`		
Description	Starts timer 1 to read if new key was pressed Debounces the read data checks the correspondence with known key ID Posts in mailbox if ID corresponds to one expected	
Arguments	`void`	
Retour	`void`	

`void KeyboardProcessSPI(void)`		
Description	Called by timer 1 interruption Processes the 5 write-read SPI cycles necessary to detect a key Stores read data in `ByteRec`	
Arguments	`void`	
Retour	`void`	

`INT8U InterpKeyKeyboard (INT8U ByteRec[])`		
Description	Converts the data received by SPI from keyboard into a character If no key is pressed, the function returns 99. If a key is pressed but no correspondence to any key is found, the function returns 0.	
Arguments	`ByteRec[]`	4 bytes received from keyboard
Retour	`Key`	ID corresponding to key pressed

Fonctions concernant le fonctionnement des LEDs du clavier :

`void UpdateLEDs(void)`		
Description	Updates the LED colors on keyboard	
Arguments	`void`	
Retour	`void`	
`void DelayLED(int cpt)`		
Description	Introduces a delay during the execution when called	
Arguments	`cpt`	How long should last the delay (`cpt` = 100 → delay = 26µs)
Retour	`void`	

`void TurnOnLEDs(void)`	
Description	Turns the keyboard LEDs on

Arguments	void	
Retour	void	

INT16U TurnOnLED (INT8U column, INT8U row, INT8U color)		
Description	Turns the selected LED on	
Arguments	column row color	Column of the selected LED [1..4] Row of the selected LED [1..4] Selected color of the selected LED [1..3] (1-red 2-blue 3-green)
Retour	val	Value to be sent to 74HC4094 a and b

void TurnOffLEDs (void)		
Description	Turns all LEDs off	
Arguments	void	
Retour	void	

Afficheur

void InitDisplay(void)		
Description	Inits the target display	
Arguments	void	
Retour	void	

void TurnOnDisplay(void)		
Description	Turns the target display on	
Arguments	void	
Retour	void	

void TurnOffDisplay(void)		
Description	Turns the target display off	
Arguments	void	
Retour	void	

void WriteDisplay(unsigned char string[])		
Description	Writes a string of characters on the target display	
Arguments	string	String of characters to be sent to display
Retour	void	

void WriteChDisplay(unsigned char ch)		
Description	Writes a character on the target display	
Arguments	ch	Character to be sent to display
Retour	void	

void ClearDisplay(void)		
Description	Clears the screen on the target display	
Arguments	void	
Retour	void	

void NewLineDisplay(void)		
Description	Sets a new line on the target display	
Arguments	void	

Retour	void	

void SaveAsStartupDisplay(void)		
Description	Saves startup screen on the target display	
Arguments	void	
Retour	void	
Note	String sent to display must have 80 (20x4) characters exactly	

void SetCursorPosDisplay(int col, int row)		
Description	Sets the cursor position on the target display	
Arguments	col row	Column chosen (1 <= col <= 20) Row chosen (1 <= row <= 4)
Retour	void	

void CursorBackDisplay(void)		
Description	Sets a new line on the target display	
Arguments	void	
Retour	void	

void InitAccentsDisplay(void)		
Description	Initializes the display special characters as é,è,à accents	
Arguments	void	
Retour	void	
Note	Do not use ID=0. Character will be displayed randomly at position (1,1) after startup screen	

unsigned char ConvertAccentsDisplay(unsigned char Data)		
Description	Converts special characters (é è à) into display special character ID	
Arguments	Data	Initial character possibly accentuated (unsigned char)
Retour	Data_acc	Accuentuated character replaced by special char ID

A.1.3 Application

int main (void)		
Description	Main program. Does initializations (variables, tasks, events...)	
Arguments	void	
Retour	int	

static void CellStatUpdtTask (void *p_arg)		
Description	Checks cell mailboxes to see if an message was sent from cell module	
Arguments	void	Task arguments pointer
Retour	void	

static void CellSendCmdTask (void *p_arg)		
Description	Sends commands from CellReqsMbox to cell module	
Arguments	void	Task arguments pointer
Retour	void	

`static void GSMCallDetectionTask (void *p_arg)`		
Description	Checks if a call is incoming and posts to `GSMMbox`	
Arguments	`void`	Task arguments pointer
Retour	`void`	

`static void KeyboardKeyTask (void *p_arg)`		
Description	Detects new key pressed ans posts to `KBDMbox`, with pointer to key ID	
Arguments	`void`	Task arguments pointer
Retour	`void`	

`static void KeyboardLedTask (void *p_arg)`		
Description	Updates LEDs color pattern	
Arguments	`void`	Task arguments pointer
Retour	`void`	

`static void KeyboardStatUpdtTask (void *p_arg)`		
Description	Checks `KBDMbox` and does necessary action of yes	
Arguments	`void`	Task arguments pointer
Retour	`void`	

`static void TimeoutTask (void *p_arg)`		
Description	Turns display and keyboard off if platform is inactive. Turns on if active again	
Arguments	`void`	Task arguments pointer
Retour	`void`	

`static void MainAppTask (void *p_arg)`		
Description	Main application task	
Arguments	`void`	Task arguments pointer
Retour	`void`	

A.2 Code pertinent

Le but de cette section n'est pas de lister tout le code relatif au projet, mais plutôt, de mentionner quelques portions dont le code est construit de façon un peu originale.

A.2.1 Code de détection de l'appui d'une touche

```
void ReadKeyboard(void) {

      Tmr_Timer1Start ();                    // Enable Timer 1
      step = 0;                              // Init Timer1 variable

      ID_old_old = ID_old;                   // Debounce : ID(n-2) = ID(n-1)
      ID_old = ID;                           // Debounce : ID(n-1) = ID(n)
      ID = InterpKeyKeyboard (ByteRec);      // Identifies newly read Key ID

      if ((ID_old_old == 99) && (ID_old == ID) && (ID >0)
            && (ID <= NB_KEYS)) {            // Is Key ID a new key ID?

            pKeys_Mbox = (pKeys+ID-1);
            OSMboxPost(KeyboardMbox,(void*)pKeys_Mbox);
      }
}
```

```
void KeyboardProcessSPI(void) {

      switch(step){
case 0:
      IOSET |= 0x0400000;      // Enable 74LS164a and 74LS165
      ReadWriteSPI(0xFE);      // WRITE #0 to scan Row 0
      step = 1;
      break;
case 1:
      IOCLR |= 0x2000000;      // Load data to be read on 74xx165 (step 1)
      step = 2;
      break;
case 2:
      IOSET |= 0x2000000;      // Load data to be read on 74xx165 (step 2)
      ByteRec[0] = ReadWriteSPI_fin();    // Dummy READ
      step = 3;
      break;
//-------------------------------------------------------------------------
```

```
case 3:
      ReadWriteSPI(0xFD);     // WRITE #1 to scan Row 1
      step = 4;
      break;
case 4:
      IOCLR |= 0x2000000;     // Load data to be read on 74xx165 (step 1)
      step = 5;
      break;
case 6:
      ReadWriteSPI(0xFB);     // WRITE #2 to scan Row 2
      step = 7;
      break;
case 7:
      IOCLR |= 0x2000000;     // Load data to be read on 74xx165 (step 1)
      step = 8;
      break;
case 8:
      IOSET |= 0x2000000;     // Load data to be read on 74xx165 (step 2)
      ByteRec[1] = ReadWriteSPI_fin();    // READ #1 to get Row 1 state
      step = 9;
      break;
//-------------------------------------------------------------------------
case 9:
      ReadWriteSPI(0xF7);     // WRITE #3 to scan Row 3
      step = 10;
      break;
case 10:
      IOCLR |= 0x2000000;     // Load data to be read on 74xx165 (step 1)
      step = 11;
      break;


case 11:
      IOSET |= 0x2000000;     // Load data to be read on 74xx165 (step 2)
      ByteRec[2] = ReadWriteSPI_fin();    // READ #2 to get Row 2 state
      step = 12;
      break;
//-------------------------------------------------------------------------
case 12:
      ReadWriteSPI(0xFF);     // Dummy WRITE
      step = 13;
      break;
case 13:
      IOCLR |= 0x2000000;     // Load data to be read on 74xx165 (step 1)
      step = 14;
      break;
case 14:
      IOSET |= 0x2000000;     // Load data to be read on 74xx165 (step 2)
      ByteRec[3] = ReadWriteSPI_fin();    // READ #3 to get Row 3 state
      IOCLR |= 0x0400000;     // Disable 74LS164a and 74LS165
      Tmr_Timer1Stop ();      // Stop Timer 1
      step = 15;
      break;
//-------------------------------------------------------------------------
default:
      break;
}}
```

A.2.2 Code de rafraîchissement des LEDs du clavier

```
void UpdateLEDs(void) {

   if (flagLEDsOff) {
        ValToSend = 0x3F3F;

        ReadWriteSPI (ValToSend >> 8);          // Send MSBs
        DelayLED(145);
        IOSET |= 0x1000000;
        DelayLED(16);
        IOCLR |= 0x1000000;
        ReadWriteSPI_fin();

        ReadWriteSPI (ValToSend & 0x00FF);      // Send LSBs
        DelayLED(145);
        IOSET |= 0x0800000;
        DelayLED(16);
        IOCLR |= 0x0800000;
        ReadWriteSPI_fin();
   }else {
        if (LEDsUpdateCnt == NB_KEYS)     // If all keys LEDs were checked
            LEDsUpdateCnt = 0;            // Skip to first key LED on kboard

        while ((pKeys+LEDsUpdateCnt)->LedOn == 0){// If present LED is off
            LEDsUpdateCnt++;              // Skip to next key On LED
            if (LEDsUpdateCnt == NB_KEYS)// If all key LEDs were checked
                LEDsUpdateCnt = 0;        // Skip to first key LED on kboard
        }
        ValToSend = TurnOnLED(      (pKeys+LEDsUpdateCnt)->Column,
                                (pKeys+LEDsUpdateCnt)->Row,
                                (pKeys+LEDsUpdateCnt)->Color );
        LEDsUpdateCnt++;                          // Update counter

        ReadWriteSPI (ValToSend >> 8);          // Send MSBs
        DelayLED(145);
        IOSET |= 0x1000000;
        DelayLED(16);
        IOCLR |= 0x1000000;
        ReadWriteSPI_fin();

        ReadWriteSPI (ValToSend & 0x00FF);      // Send LSBs
        DelayLED(145);
        IOSET |= 0x0800000;
        DelayLED(16);
        IOCLR |= 0x0800000;
        ReadWriteSPI_fin();
   }
}
```

A.2.3 Code d'allumage d'une LED (construction trame SPI)

```
INT16U TurnOnLED  (INT8U column, INT8U row, INT8U color){
   INT16U val = 0x3F3F;

   switch (column){
        case(1): val |= 0x8000; val &= 0xBF3F; break;
        case(2): val |= 0x4000; val &= 0x7F3F; break;
        case(3): val |= 0x0080; val &= 0x3FBF; break;
        case(4): val |= 0x0040; val &= 0x3F7F; break;
   }
   switch (row){
        case(1): if (color == 1)    { val |= 0x3F3E; val &= 0xFFFE; }
             else if (color == 2)   { val |= 0x3F3D; val &= 0xFFFD; }
             else if (color == 3)   { val |= 0x3F3B; val &= 0xFFFB; } break;
        case(2): if (color == 1)    { val |= 0x3F37; val &= 0xFFF7; }
             else if (color == 2)   { val |= 0x3F2F; val &= 0xFFEF; }
             else if (color == 3)   { val |= 0x3F1F; val &= 0xFFDF; } break;
        case(3): if (color == 1)    { val |= 0x3E3F; val &= 0xFEFF; }
             else if (color == 2)   { val |= 0x3D3F; val &= 0xFDFF; }
             else if (color == 3)   { val |= 0x3B3F; val &= 0xFBFF; } break;
        case(4): if (color == 1)    { val |= 0x373F; val &= 0xF7FF; }
             else if (color == 2)   { val |= 0x2F3F; val &= 0xEFFF; }
             else if (color == 3)   { val |= 0x1F3F; val &= 0xDFFF; } break;
   }
   return val;
}
```

A.3 Schéma électrique de l'ensemble du montage

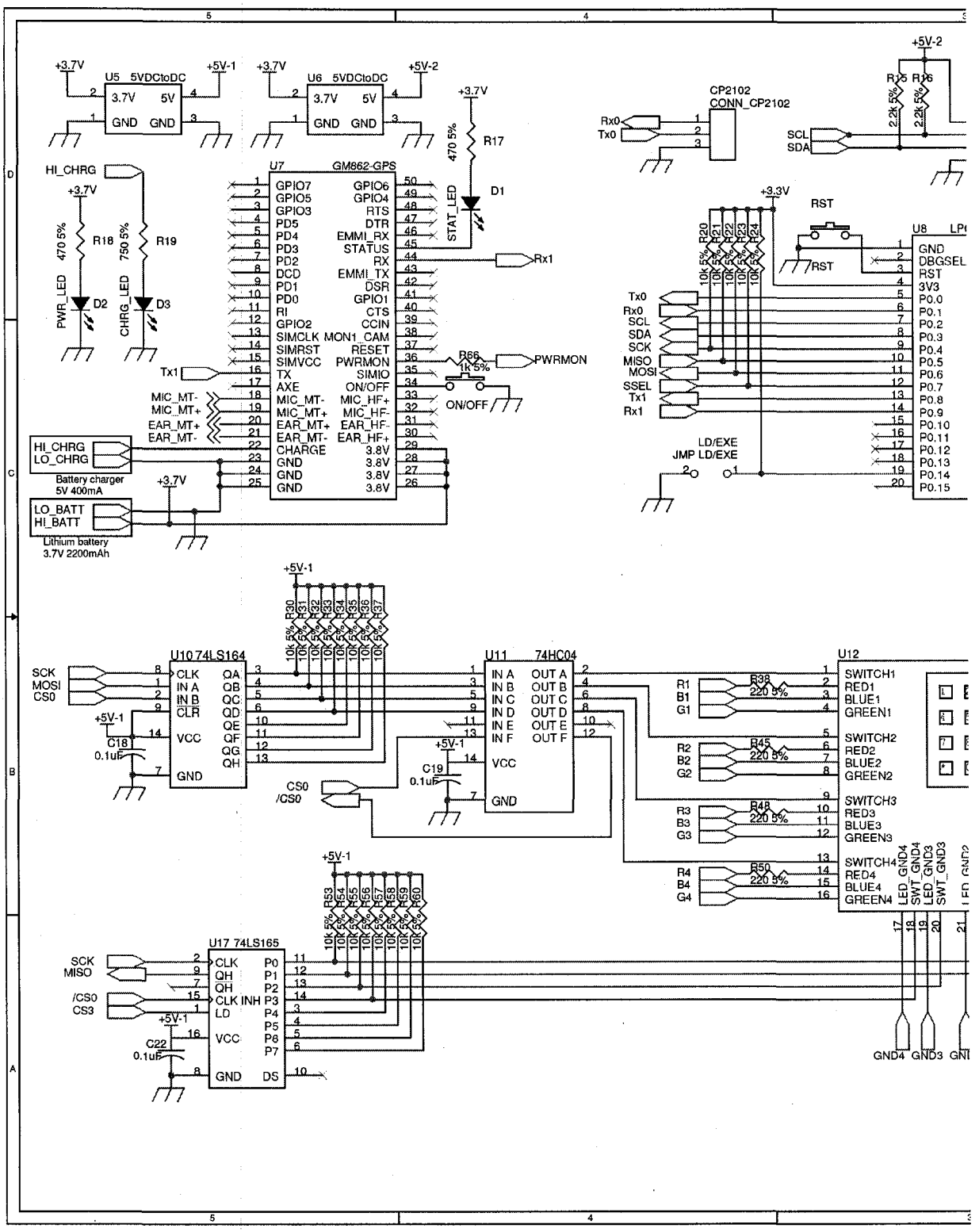


Figure A-1 – Schéma électrique global – Microcontrôleur et périphériques – 1

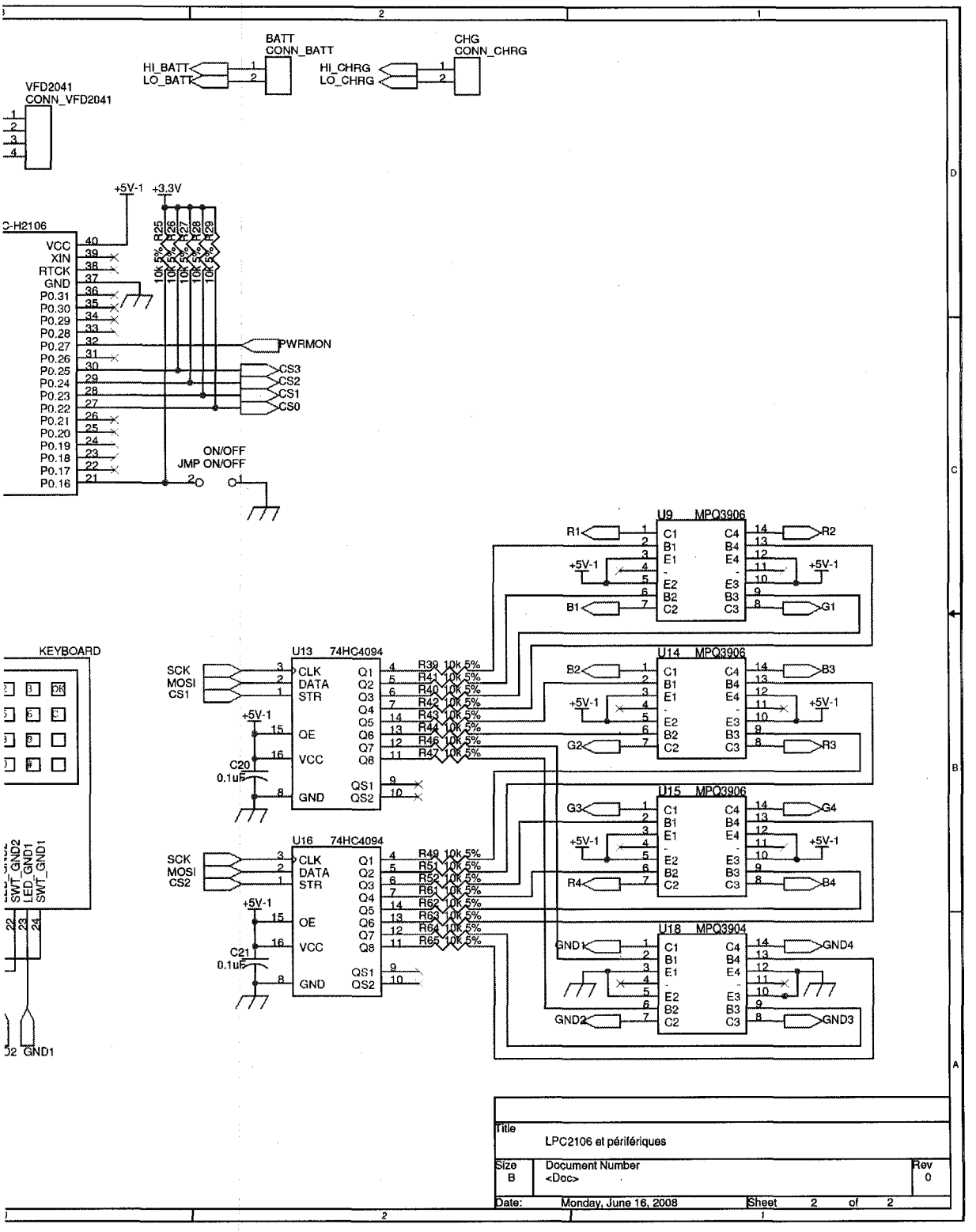


Figure A-2 – Schéma électrique global – Microcontrôleur et périphériques – 2

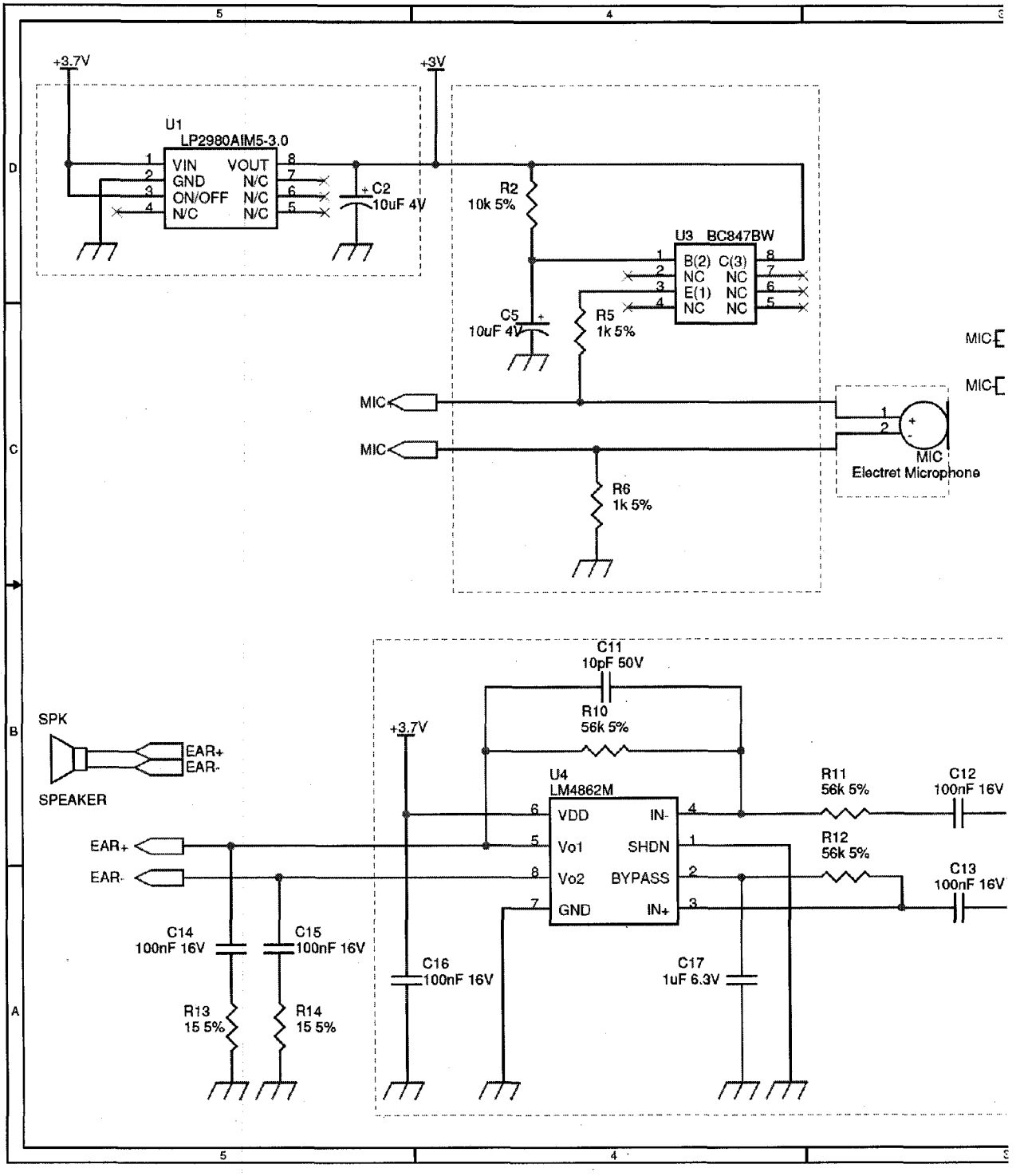


Figure A-3 – Schéma électrique global – GM862 et interface audio – 1

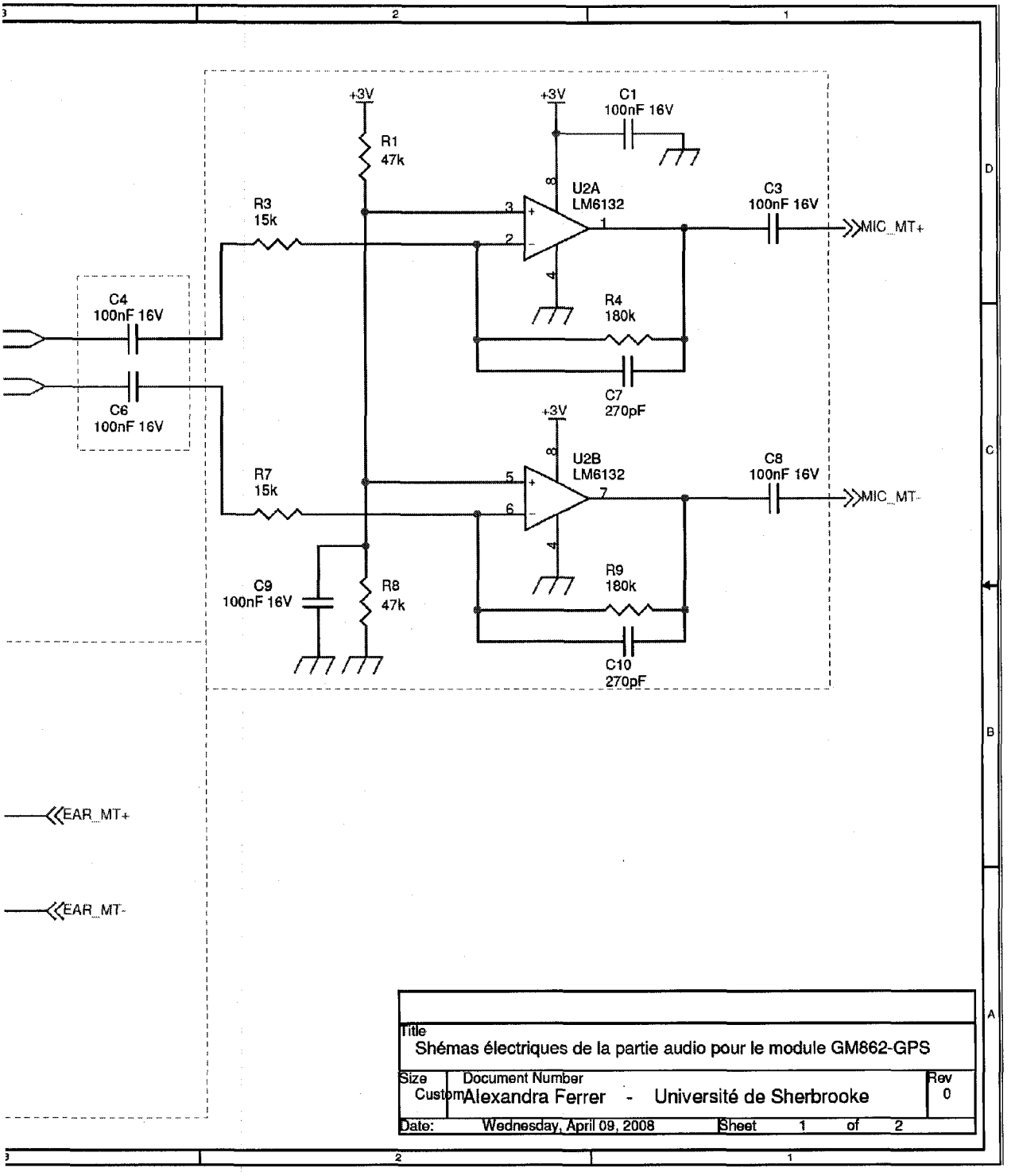


Figure A-4 – Schéma électrique global – GM862 et interface audio – 2

BIBLIOGRAPHIE

[FRUCHTERMAN, 2003] FRUCHTERMAN, J. R. (octobre 2003). « In the Palm of Your Hand: A Vision of the Future of Technology for People with Visual Impairments», Journal of Visual Impairment & Blindness, vol. 97, n° 10.

[GILL, 1996] GILL, J. (1996). « Telecommunications: The Missing Links for People with Disabilities », The European Commission - Directorate-General XIII - Telecommunications, Information Market and Exploration of Research, [en ligne], [http://www.empowermentzone.com/misslink.txt] (consulté le 29 mai 2008)

[KAASINEN, 2003] KAASINEN, E. (2003). « User needs for location-aware mobile services », Pers Ubiquit Comput, vol. 7, n°1, p.70-79

[KOESTER, 2006] KOESTER, H.H. (Printemps 2006). « Factors that influence the performance of experienced speech recognition users », Assistive Technology, vol. 18, n° 1, p.56-76

[LABROSSE, 2002] LABROSSE, J.J. (15 juin 2002). MicroC/OS-II The Real-Time Kernel, Lawrence KS USA, Second Edition, CMP Books, 648 p.

[MO, 2006] MATRIX ORBITAL (26 Juin 2006). « VFD2041 Technical Manual », rév. 2.0, [en ligne], [http://www.matrixorbital.ca/manuals/LCDVFD_series/VFD204] (consulté le 20 mai 2008)

[MUELLER et coll., 2005] MUELLER, J., JONES, M., BRODERICK, L., HABERMAN, V. (Printemps 2005). « Assessment of user needs in wireless technologies », Assistive Technology, vol. 17, n° 1, p.57-71

[NGUYEN et coll., 2001] NGUYEN, T., DOWNING, A., GARRETT, R. (2001). « Making mobile phones more accessible », ARATA Conference Paper, [en ligne], [http://www.novitatech.org.au/library/Making_mobile_phones_accessible.pdf] (consulté le 4 septembre 2008)

[NGUYEN et coll., 2006] NGUYEN, T., GARRETT, R., DOWNING, A., WALKER, L., HOBBS, D. (mars 2006). « Telecommunications access - Matching available technologies to people with physical disabilities », Australasian Physical and Engineering Sciences in Medicine, vol. 29, n° 1, p.87-97

[NXP, 2006ds] NXP (25 juillet 2006), « LPC2104/2105/2106 Product data sheet », rév. 06, [en ligne], [http://www.standardics.nxp.com/products/lpc2000/datasheet/lpc2104.lpc2105.lpc2106.pdf] (consulté le 29 mai 2008)

[NXP, 2003um] NXP (2 octobre 2003), « LPC2106/2105/2104 User Manual », rév. preliminary, [en ligne], [http://www.nxp.com/acrobat_download/usermanuals/UM_LPC2106_2105_2104_2.pdf] (consulté le 29 mai 2008)

[SMITH-JACKSON et coll.,2003] SMITH-JACKSON, T. L., NUSSBAUM, M. A., MOONEY, A. M. (20 mai 2003). « Accessible cell phone design: development and application of a needs analysis framework », Disability and Rehabilitation, vol. 25, n° 10, p.549–560

[SPARKFUN, 2008] SPARKFUN (7 janvier 2008), « How to use the VPack_PCB board », rév. 1.1, [en ligne], [http://www.sparkfun.com/datasheets/Prototyping/Batteries/VPack_PCB_v1r1_general.pdf] (consulté le 20 août 2008)

[TELIT, 2007hw] TELIT (20 septembre 2007), « GM862-GPS Hardware User Guide », rév. 8, réf. 1vv0300728, [en ligne], [http://www.gm862.com/module/infopool/download.php?id=871] (consulté le 29 mai 2008)

[TELIT, 2007sw] TELIT (5 juillet 2007), « GM862-GPS Software User Guide », rév. 5, réf. 1vv0300729, [en ligne], [http://www.gm862.com/module/infopool/download.php?id=533] (consulté le 29 mai 2008)

[TELIT, 2008pd] TELIT (11 mars 2008), « GM862 Product Description », rév. 9, réf. 80272ST10019a, [en ligne], [http://www.gm862.com/module/infopool/download.php?id=165] (consulté le 29 mai 2008)

[TELIT, 2007at] TELIT (19 novembre 2007), « AT Commands Reference Guide, SW Release 7.02.04 for GM862-QUAD, GM862-QUAD-PY, GE862-GPS, GE863-QUAD, GE863-PY, GE863-GPS, GE864-QUAD, GE864-PY, GC864-QUAD and GC864-PY », rév. 4, réf. 80000ST10025a, [en ligne], [http://www.gm862.com/module/infopool/download.php?id=542] (consulté le 3 juin 2008)